ECONSTOR

WWW.ECONSTOR.EU

Der Open-Access-Publikationsserver der ZBW – Leibniz-Informationszentrum Wirtschaft
The Open Access Publication Server of the ZBW – Leibniz Information Centre for Economics

Christ, Florian; Riemer, Kai

Working Paper

Das Beergame: Realisierung einer Softwarevariante für den Einsatz in E-Commerce-Lehrveranstaltungen

Internetökonomie und Hybridität, No. 54

Provided in Cooperation with:
European Research Center for Information Systems (ERCIS), University of Münster

Suggested Citation: Christ, Florian; Riemer, Kai (2008) : Das Beergame: Realisierung einer Softwarevariante für den Einsatz in E-Commerce-Lehrveranstaltungen, Internetökonomie und Hybridität, No. 54

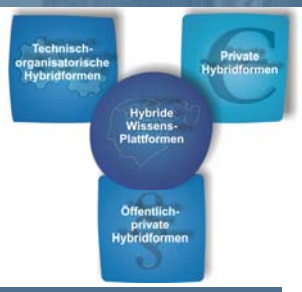
This Version is available at:
http://hdl.handle.net/10419/46596

Nutzungsbedingungen:
Die ZBW räumt Ihnen als Nutzerin/Nutzer das unentgeltliche, räumlich unbeschränkte und zeitlich auf die Dauer des Schutzrechts beschränkte einfache Recht ein, das ausgewählte Werk im Rahmen der unter
→ http://www.econstor.eu/dspace/Nutzungsbedingungen
nachzulesenden vollständigen Nutzungsbedingungen zu vervielfältigen, mit denen die Nutzerin/der Nutzer sich durch die erste Nutzung einverstanden erklärt.

Terms of use:
The ZBW grants you, the user, the non-exclusive right to use the selected work free of charge, territorially unrestricted and within the time limit of the term of the property rights according to the terms specified at
→ *http://www.econstor.eu/dspace/Nutzungsbedingungen*
By the first use of the selected work the user agrees and declares to comply with these terms of use.

ZBW Leibniz-Informationszentrum Wirtschaft
Leibniz Information Centre for Economics

Mitglied der Leibniz-Gemeinschaft



Prof. Dr. Dieter Ahlert, PD Dr. Detlef Aufderheide, Prof. Dr. Klaus Backhaus,
Prof. Dr. Jörg Becker, Prof. Dr. Heinz Lothar Grob, Prof. Dr. Karl-Hans Hartwig,
Prof. Dr. Thomas Hoeren, Prof. Dr. Heinz Holling, Prof. Dr. Bernd Holznagel,
Prof. Dr. Stefan Klein, Prof. Dr. Thomas Langer, Prof. Dr. Andreas Pfingsten

Internetökonomie und Hybridität

Nr. 54

FLORIAN CHRIST, KAI RIEMER

Das Beergame
Realisierung einer Softwarevariante für den Einsatz in E-Commerce-Lehrveranstaltungen

European Research Center
for Information Systems

Westfälische
Wilhelms-Universität
Münster

Gefördert durch:

Förderkennzeichen:
01 AK 704

Projektträger:

INTERNETÖKONOMIE

Internetökonomie und Hybridität

Sprecher
Prof. Dr. Heinz Lothar Grob

www.ercis.org

NR. 54

Florian Christ, Kai Riemer

Das Beergame

Realisierung einer Softwarevariante für den Einsatz in E-Commerce-Lehrveranstaltungen

KOMPETENZCENTER INTERNETÖKONOMIE UND HYBRIDITÄT
SPRECHER: PROF. DR. HEINZ LOTHAR GROB, KOORDINATOR: CHRISTIAN BUDDENDIECK,
LEONARDO-CAMPUS 3, 48149 MÜNSTER, TEL. (0251) 83-38000, FAX. (0251) 83-38009
EMAIL: GROB@UNI-MUENSTER.DE
http://www-wi.uni-muenster.de/aw/

Februar 2008

Inhaltsverzeichnis

Inhaltsverzeichnis II
Abbildungsverzeichnis IV
1 Einleitung 1
2 Das Beergame zur Simulation des Bullwhip-Effekts 3
2.1 Der Bullwhip-Effekt 3
2.2 Entstehung des Beergame 4
2.3 Regeln und Ablauf des klassischen Brettspiels 5
2.4 Alternative Variante des Brettspiels 8
2.5 Konsequenzen für eine softwaretechnische Realisierung 10
3 Anforderungsanalyse zur softwaretechnischen Realisierung 12
3.1 Ermittlung des Funktionsumfangs 12
3.1.1 Anwendungsfälle des Spielleiters 13
3.1.2 Anwendungsfälle des Spielers 16
3.2 Technische Anforderungen 18
4 Objektorientierte Modellierung des Datenmodells 19
5 Softwarearchitektur 24
5.1 Erarbeitung der Softwarearchitektur 24
5.2 Auswahl der Techniken zur Umsetzung der Architektur 26
5.2.1 Technologien für das Frontend 26
5.2.1.1 Serverseitiges Skript und Servlets 26
5.2.1.2 Rich Internet Applications 28
5.2.1.3 Vergleich der Alternativen und Auswahl 32
5.2.2 Datenaustausch und persistente Datenhaltung 35
5.2.3 Technologien für das Backend 37
6 Implementierung 41
6.1 Backend 41
6.1.1 XML-Objekte 42
6.1.2 Anfragen an die Controller 43
6.1.3 Exception-Handling 45
6.1.4 Export der Spieldaten als Excel-Spreadsheet 46
6.2 Frontend 47
6.2.1 Verbindung zum Server und Aktualisierung in Echtzeit 47
6.2.2 Verwendung der XML-Daten 49
6.2.3 Mehrsprachigkeit 51
6.2.4 Diagramme 52
7 Mögliche Erweiterungen 55
7.1 Alternatives Frontend 55
7.2 Alternativer Zugriff auf das Beergame Backend 56
7.3 Externer Zugriff auf gespeicherte Daten des Beergame 56
7.4 Erweiterung der Programmlogik 57
8 Bedienung der Software 58
8.1 Benutzerschnittstelle des Spielleiters 58

8.1.1 Veranstaltung auswählen....58
8.1.2 Veranstaltung anlegen / Spielkonfiguration festlegen....59
8.1.3 Spieler zu Spielen zuordnen....61
8.1.4 Alle Spiele beobachten....62
8.1.5 Detailansicht eines Spiels....63
8.1.6 Nachrichten senden / empfangen....64
8.2 Benutzerschnittstelle des Spielers....66
8.2.1 Veranstaltung betreten....66
8.2.2 Spielansicht I (Tabelle)....66
8.2.3 Spielansicht II (Animation)....67
8.2.4 Nachrichten senden / empfangen....70
9 Fazit nach der Entwicklung....71
Literaturverzeichnis....72
Anhang....74
A XML Datenstruktur....74
A.a Event 74
A.b Admin 75
A.c Game 75
A.d Player 76
A.e Message....76
A.f RoundData....77
A.g EventPreferences....78
B Mögliche Anfragen an die Controller....80
B.a AdminController....80
B.b PlayerController....81
C Benutzte externe Komponenten....81

Abbildungsverzeichnis

Abb. 3.1: Steigende Variabilität der Bestellungen in der Supply-Chain 4
Abb. 3.2: Spielbrett des Beergame in Ausgangsstellung 6
Abb. 3.3: Alternative Variante des Brettspiels 9
Abb. 4.1: Akteure des Beergame in UML-Notation 12
Abb. 4.2: Anwendungsfalldiagramm des Spielleiters 16
Abb. 4.3: Anwendungsfalldiagramm des Spielers 17
Abb. 5.1: Klassendiagramm vor der Implementierungsphase 23
Abb. 6.1: Client/Server Architektur der Beergame-Software 25
Abb. 6.2: Erweiterte Softwarearchitektur 40
Abb. 7.1: Paketstruktur der Java-Klassen des Backend 41
Abb. 7.2: URL zur Abgabe einer Bestellung 43
Abb. 7.3: Variationen des Liniendiagramms 54
Abb. 9.1: Übersicht der laufenden Veranstaltungen 58
Abb. 9.2: Bildschirm zur Festlegung der Spielkonfiguration 59
Abb. 9.3: Bildschirm zur Zuordnung von Spielern zu Spielen 62
Abb. 9.4: Bildausschnitt der Spielübersichtsmaske 63
Abb. 9.5: Bildschirm zur detaillierten Beobachtung eines Spiels 64
Abb. 9.6: Fenster zum Senden und Empfangen von Nachrichten 65
Abb. 9.7: Anmeldebildschirm 66
Abb. 9.8: Bildausschnitt der Tabellenansicht 67
Abb. 9.9: Animationsansicht 68

1 Einleitung

In Handels- und Produktionsunternehmen nimmt das Supply-Chain-Management (SCM) eine immer wichtigere Rolle ein, da durch eine optimale Ausgestaltung der Lieferkette von der Produktion bis hin zum Kunden Wettbewerbsvorteile geschaffen werden können. Durch eine überbetriebliche Optimierung der logistischen und organisatorischen Prozesse sowie der Informationsinfrastruktur soll erreicht werden, das Angebot und die Nachfrage an Produkten innerhalb der Supply-Chain aufeinander abzustimmen, um Überproduktionen und erhöhte Lagerkosten zu vermeiden.[1] Ein Phänomen, das insbesondere in unkoordinierten Lieferketten auftritt und recht deutlich die Notwendigkeit des Supply-Chain-Management zeigt, ist der sog. „Bullwhip-Effekt". Verwaltet jedes Unternehmen der Supply-Chain seinen Lagerbestand autonom, so führt dies zu Schwankungen in den Bestellmengen. Diese Schwankungen verstärken sich zunehmend, je weiter man sich entlang der Supply-Chain vom Kunden bis bin zur Produktion bewegt. Die Folge sind hohe Kosten, die aus großen Sicherheitsbeständen oder nicht erfüllbaren Aufträgen resultieren.

Mit dem *Beergame* existiert ein einfaches und anschauliches Instrument, welches es ermöglicht, spielerisch innerhalb einer (Lern-)Gruppe die Material- und Informationsflüsse in Supply-Chains zu simulieren und so die Ursachen und Wirkungen des Bullwhip-Effekts verstehen und analysieren zu lernen. Seit seiner Einführung in den 60er Jahren des vergangenen Jahrhunderts erfreut sich das Beergame daher großer Beliebtheit und wird häufig in Seminaren oder Workshops zum Thema Supply-Chain-Management oder E-Commerce angewendet. Dabei existieren mittlerweile eine ganze Reihe verschiedener Umsetzung des Beergame: als Brettspiel, als Rollenspiel oder als Softwaresimulation. Die praktische Durchführung der klassischen Variante des Beergame als Brettspiel kann jedoch schnell zu Problemen führen. Beispielsweise unterlaufen den Spielern hierbei während des Spiels gelegentlich Rechenfehler, wodurch die Aussagekraft des Spiels und seiner Ergebnisse beeinträchtigt wird. Auch ist die Verwendung des Brettspiels meist nur in kleineren Gruppen praktikabel, da mit steigender Teilnehmerzahl der Bedarf an Platz und an Koordination nicht mehr ausreichend gedeckt werden kann.

Der Einsatz von Informationstechnik kann zur Lösung der angesprochenen Probleme beitragen. Überlässt man die Berechnungen einer Software, können fehlerhafte Spielstände vermieden werden. Zudem kann die Nutzung von Netzwerktechnik die örtliche Bindung der Spieler aufheben. Wichtig hierbei ist es jedoch, den Rollenspielcharakter des Spiels zu erhalten, um die unmittelbare Erfahrbarkeit der Interaktionen mit den anderen Supply-Chain-Mitgliedern zu erhalten. Eine reine Softwaresimulation ist hierfür jedoch nicht ge-

[1] Vgl.. Schulte (2005), S.525 ff.

eignet. Ziel des in diesem Bericht vorgestellten Projektes war es daher, eine Software zu entwickeln, mit der das Beergame alternativ zum klassischen Brettspiel durchgeführt werden kann. Diese sollte an dem sehr einfach gehaltenen Konzept des Beergame festhalten, einen Teil der Haptik auf den Bildschirm transferieren, dabei jedoch die Unzulänglichkeiten des Brettspiels überwinden.

Der vorliegende Bericht zeichnet hauptsächlich den Entwicklungsprozess der softwaretechnischen Umsetzung des Beergame nach. In der gleichen Herausgeberreihe gibt es darüber hinaus einen einführenden Arbeitsbericht zu den Grundlagen der Bullwhip-Effekts und seinen Ursachen, sowie zur Rolle des Beergame zur Simulation des Effekts. Dort werden insbesondere auch Maßnahmen aus Supply-Chain-Management und E-Commerce zur Bekämpfung der Supply-Chain-Koordinationsprobleme diskutiert.

Im Weiteren werden zunächst zur Einführung in die Problemstellung die Ursprünge des Beergame als Spiel zur praktischen Nachstellung des Bullwhip-Effekts nachgezeichnet (Kapitel 2). Für zwei Brettspielvarianten des Beergame werden in diesem Kapitel Spielregeln und Spielablauf beschrieben sowie Problemfelder identifiziert. Das dritte Kapitel beinhaltet eine detaillierte Anforderungsanalyse für die spätere softwaretechnische Umsetzung. Die Entwicklung der Software beginnt mit der Erstellung eines objektorientierten Datenmodells für das Beergame in Kapitel 4. Kapitel 5 beinhaltet zunächst die Erarbeitung einer angemessenen Softwarearchitektur, anschließend wird ausführlich der Entscheidungsprozess bezüglich der bei der Umsetzung zu verwendenden Technologien der Softwareentwicklung nachgezeichnet. Schließlich wird in Kapitel 6 die Lösung zentraler Herausforderungen beschrieben, die sich während der Implementierung der Software ergaben. Der Bericht schließt mit einem Ausblick auf mögliche zukünftige Erweiterungen der Software, zu deren Umsetzung bereits während der Entwicklung zahlreiche Schnittstellen vorgesehen wurden. Schließlich beinhaltet Kapitel 8 eine detaillierte Beschreibung der Benutzerschnittstelle der Software und kann während der Nutzung als Benutzerhandbuch verwendet werden.

2 Das Beergame zur Simulation des Bullwhip-Effekts

2.1 Der Bullwhip-Effekt

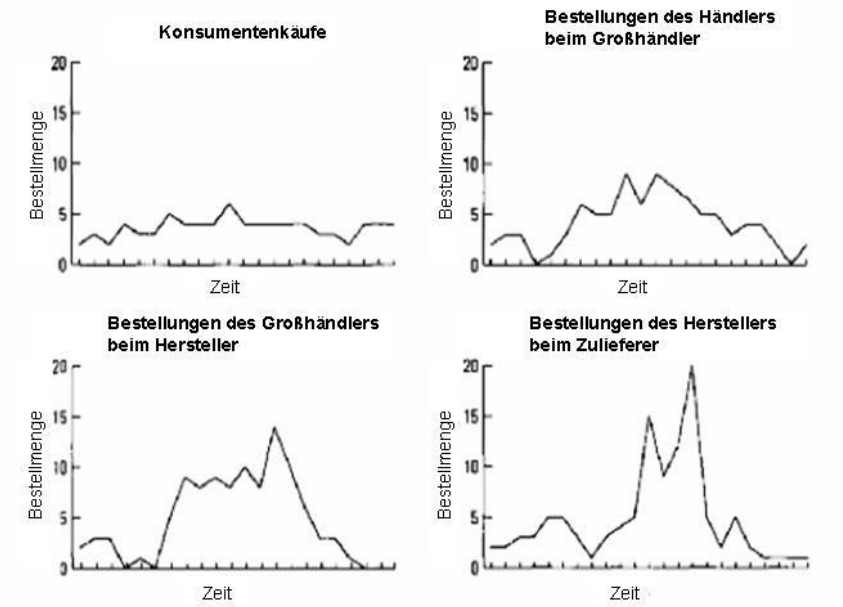
In mehrstufigen Lieferketten lässt sich beobachten, dass trotz relativ geringer Nachfragevariabilität auf der Endkundenseite sowohl Bestellmengen wie auch Lagerbestände auf den höheren Stufen der Lieferkette großen Schwankungen unterliegen. So führen aufgrund von Störungen und Verzerrungen bei der Übermittlung des Bedarfs bereits kleine Änderungen der Endkundennachfrage stromaufwärts in der Lieferkette zu immer größeren Ausschlägen in den Bestellmengen.[2] Dieses Phänomen ist allgemein als Bullwhip-Effekt, in einigen Wirtschaftszweigen auch als Whiplash- oder Whipsaw-Effekt, bekannt.[3] Als Konsequenz dieser Koordinationsprobleme ergeben sich unter anderem erhöhte Lagerbestände, ineffiziente Bedarfsplanung und schlechte Kapazitätsauslastung, die wiederum zu höheren Kosten und schlechtem Service führen können.[4]

Erstmals beschrieben wurde dieses Phänomen durch FORRESTER (1961), der in einem Simulationsmodell den Zusammenhang zwischen Bestellungen und Lagerbeständen in mehreren Fallstudien erforschte.[5] Als Ursache für das Aufschaukeln der Nachfragevariabilität nannte er industriedynamische Prozesse und im Zeitablauf unkonstantes Verhalten industrieller Organisationen und er wies erstmals auf die Wichtigkeit der ganzheitlichen Betrachtung des Systems hin.[6] Der Begriff Bullwhip-Effekt wurde zu Beginn der 1990er Jahre des letzten Jahrhunderts geprägt, als verschiedene Firmen den Effekt erstmals in der Praxis untersuchten. Der Konsumgüterhersteller Procter&Gamble (P&G) beobachtete bei einer Marktuntersuchung für sein Produkt Pampers-Windeln, dass die Verkaufszahlen bei den Endhändlern im Zeitablauf erwartungsgemäß relativ konstant waren[7]. Bei der weiteren Betrachtung der Lieferkette stellte man fest, dass die Nachfrage der Zwischenhändler bereits stärker variierte und die eigenen Rohstoffbestellungen bei den Zulieferfirmen erheblichen Schwankungen unterworfen war. Hewlett-Packard (HP) fand heraus, dass sich die Absatzschwankungen bei Druckern innerhalb der Supply-Chain verstärkten und die Kapazitäts- und Produktionsplanung erschwert wurden, da eine Unterscheidung zwischen echter und fiktiver Variabilität des Marktbedarfs erheblich verkompliziert wurde.[8] Der italienische Nudelhersteller Barilla sah sich bei einem seiner lokalen Verteilerlager wöchentlichen

---

2 Vgl. Weber (2001).
3 Vgl. Lee, Padmanabhan, Whang (1997a), S. 93.
4 Vgl. Günthner (2005).
5 Vgl. Warburton (2004), S.150.
6 Vgl. Lee, Padmanabhan, Whang (1997b), S. 547; Forrester (1958), S. 37-66; vertiefend: vgl. Forrester (1961).
7 Vgl. Alicke, (2005), S. 99.
8 Vgl. Lee, Padmanabhan, Whang (1997a), S. 93.

Lieferungen gegenüber, die im Jahresablauf um den Faktor 70 schwankten, während sich die Lieferungen an die Supermärkte nur um weniger als den Faktor 3 veränderten.[9] Auch in der Textil- und Computerbranche konnte dieser Effekt in zahlreiche Fallstudien beobachtet werden.[10] Ein erster mathematisch-formaler Ansatz zur Erklärung des Bullwhip-Effektes und seiner zentralen Ursachen, auf die im Folgenden noch explizit eingegangen wird, findet sich bei LEE, PADMANABHAN, WHANG (1997b). In der folgenden Abbildung ist die steigende Variabilität der Bestellungen auf den verschiedenen Stufen einer vierstufigen Supply-Chain bestehend aus Konsument, Händler, Produzent und Zulieferer dargestellt.



Quelle: Lee, Padmanabhan, Whang (1997a), S. 94.

Abb. 2.1: Steigende Variabilität der Bestellungen in der Supply-Chain

2.2 Entstehung des Beergame

Das Beergame (oder auch Beer Distribution Game) wurde bereits in den 1960er Jahren am Massachusetts Institute for Technology (MIT) in einer Forschungsgruppe um Prof. Jay Wright Forrester erarbeitet.[11] Sinn des Spiels ist es, in einer nachgestellten Supply-Chain den Bullwhip-Effekt zu reproduzieren. Dabei wird den Spielern explizit aufgetragen diesen zu verhindern, was ihnen unter den nachfolgend genannten Spielvoraussetzungen allerdings nicht gelingen wird.

9 Vgl. Simchi-Levi, Kaminsky, Simchi-Levi (2003), S. 91ff.
10 Vgl. Chopra, Meindl (2001), S. 360 f.; vertiefend: siehe Hammond (1994).
11 Vgl. Sterman (1992), S.40

Es wird eine vierstufige Supply-Chain simuliert, in der von der Produktion über einen Distributor, einen Großhändler und schließlich einem Einzelhändler ein Produkt zum Verbraucher gelangen soll. Bei dem angebotenen Produkt muss es sich nicht zwingend um Bier handeln, jedoch sollte ein Produkt gewählt werden, bei dem von maximalen Lagerzeiten abstrahiert werden und welches zudem in einer festen Verpackungseinheit angeboten werden kann. Beim ursprünglichen Beergame wird das Bier in Kästen vertrieben.

Im Spiel wird jedem Spieler das Management eines der vier Unternehmen der Supply-Chain übertragen. Ziel jedes Spielers ist es, die eingehende Nachfrage bestmöglich durch den eigenen Lagerbestand zu befriedigen, ohne dabei jedoch den Sicherheitsbestand zu groß werden zu lassen. Der Lagerbestand kann durch Bestellungen bei der nachfolgenden Stufe der Supply-Chain verwaltet werden. Die Bestellungen eines Spielers stellen damit gleichzeitig die eingehende Nachfrage im nachfolgenden Unternehmen dar. Der Einzelhändler erhält als Input die Endkundennachfrage, welche die einzige externe Einflussgröße des Spiels darstellt.

Da beim Beergame der Fokus ausschließlich auf den Bullwhip-Effekt gelegt werden soll, weist die simulierte Supply-Chain einen sehr hohen Abstraktionsgrad auf. Einzige Größen von Belang sind Bestände und Liefermengen, sowie Kosten, die für Lagerhaltung oder Konventionalstrafen anfallen. Es gibt keinerlei Höchstgrenzen für Lager oder Transportmengen, auch können die Waren im Lager nicht verfallen. Zudem werden bei den Bestellungen keine Preise veranschlagt. Die anfallenden Kosten dienen ausschließlich der Abwägung des optimalen Lagerbestands.

Der Verlauf des Bullwhip-Effekts kann über die Endkundennachfrage gesteuert werden. Im vorherigen Kapitel wurde jedoch deutlich, dass der Effekt selbst bei nahezu konstantem Verlauf der Endkundennachfrage entstehen kann. Daher wird beim Beergame zumeist ein Nachfrageverlauf gewählt, der über das gesamte Spiel konstant ist, abgesehen von einem sprunghaften Anstieg in einer frühen Phase des Spiels. Dieser Sprung soll als Auslöser des Bullwhip-Effekts dienen.

2.3 Regeln und Ablauf des klassischen Brettspiels

Dreh- und Angelpunkt des Spiels ist das Spielbrett (Abb. 3.1). Das Spielbrett stellt die zu simulierende Supply-Chain stilisiert dar. In der unteren Hälfte des Bretts wird der Materialfluss zwischen den Stufen der Supply-Chain von rechts nach links, in der Oberen der Informationsfluss von links nach rechts dargestellt. Seitens des Materials gibt es für jede Stufe ein Feld, das den aktuellen Lagerbestand anzeigt. Zwischen den Lagerfeldern befinden sich jeweils zwei Felder *Shipping Delay*. Güter, die sich in einem der *Shipping Delay*

Felder befinden, wurden bereits von der vorgelagerten Stufe versendet und befinden sich auf dem Weg zum Empfänger. Die Brauerei benötigt zur Herstellung des Gutes eine gewisse Zeit. Auf den Feldern *Production Delay* werden Produkte abgelegt, die sich noch in der Produktion befinden. Seitens des Informationsflusses befinden sich in der oberen Hälfte des Spielbretts Felder zur Verwaltung der Bestellungen. Jede Stufe der Supply-Chain besitzt ein Feld, auf dem alle unbearbeiteten eingegangenen Bestellungen abgelegt werden, und eines, auf dem sich die eigene Bestellung neuer Güter befindet. Der Einzelhändler bekommt seine Aufträge nicht von einem Mitspieler, sondern direkt vom Endkunden. Die Endkundennachfrage befindet sich auf dem Feld *Customer Orders*.

Vor Beginn des Spiels wird zunächst das Spielbrett in die Ausgangsstellung gebracht. Die Ausgangsstellung ist eine ausgeglichene Supply-Chain, also eine Situation, in der alle eingehenden Bestellungen erfüllt werden können und die Lagerbestände nur um den Wert der ausgehenden Lieferungen aufgefüllt werden. Dazu werden auf jedes Lagerfeld 12 Spielsteine gelegt sowie 4 auf das jeweils linke *Shipping Delay* Feld. Ein Spielstein repräsentiert einen Kasten Bier. Auch sämtliche Bestellungsfelder werden mit einem Wert von 4 initialisiert, indem die Bestellmenge auf einem Zettel notiert und dieser verdeckt auf das jeweilige Feld gelegt wird. Auf dem Feld für die Endkundennachfrage wird ein Zettelstapel platziert, in dem für jede Runde ein Zettel mit der jeweiligen Nachfrage existiert. Auch die Endkundennachfrage startet bei einem Wert von 4 Einheiten.

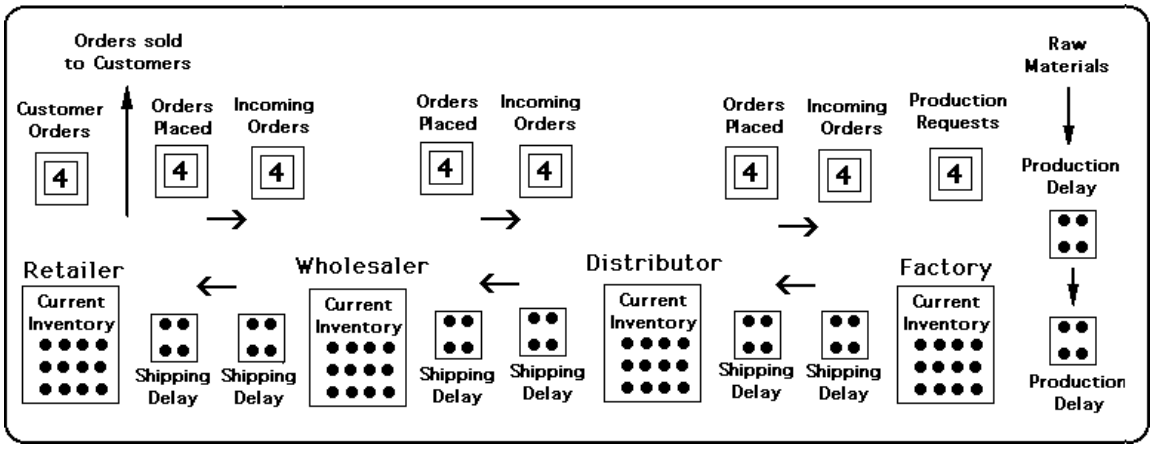


Abb. 2.2: Spielbrett des Beergame in Ausgangsstellung[12]

Bevor das Spiel beginnen kann, müssen noch die Spieler auf die einzelnen Stufen der Supply-Chain aufgeteilt werden. Jede Stufe muss mit mindestens einem Spieler besetzt sein. Weist man einer Stufe mehrere Spieler zu, so spielen diese im Team und treffen gemeinsame Entscheidungen.

12 Sterman (1992), S.40

Das Spiel läuft rundenbasiert ab, wobei eine Spielrunde etwa die Vorgänge einer Woche abbilden soll. In jeder Runde trifft vom links gelegenen Spieler eine neue Bestellung ein. Aufgabe des Spielers ist es, den eigenen Lagerbestand derart zu verwalten, dass eingehende Bestellungen möglichst zuverlässig erfüllt werden, ohne jedoch dabei den Lagerbestand zu groß werden zu lassen. Dazu kann der Spieler in jeder Runde selbst neue Ware nachbestellen. Ziel des Spiels ist es, die Gesamtkosten des Spiels zu minimieren, die sich aus Lagerkosten sowie Kosten für nicht erfüllte Bestellungen ergeben. Daher werden meist mehrere Spiele parallel ausgetragen, um nachher die Gesamtkosten der einzelnen Spiele vergleichen zu können.

Eine Spielrunde besteht aus mehreren Aktionen, die nacheinander ausgeführt werden müssen:[13]

1. Die eingehende Bestellung wird aufgedeckt. Kann die Bestellung durch den Lagerbestand gedeckt werden, so werden die bestellte Menge an Spielsteinen auf das *Shipping Delay* Feld links vom Lager gelegt. Der Einzelhändler liefert direkt an den Endkunden, die betreffenden Spielsteine werden aus dem Spiel genommen. Danach kann der Bestellzettel vom Brett entfernt werden. Sind zu wenige Spielsteine im Lager, so wird die Bestellung so weit wie möglich erfüllt. Auf dem Bestellzettel wird die noch ausstehende Menge notiert und der Zettel wieder zurück an seinen Platz gelegt. Diese Fehlmengen werden nachgeliefert, sobald sich wieder Spielsteine im Lager befinden. Bei der Brauerei können keine Fehlmengen auftreten. Sie bekommt immer die auf dem Feld *Production Request* geforderte Menge an Spielsteinen und legt diese auf das oberste *Production Delay* Feld.

2. Die Spielsteine aus dem *Shipping Delay* Feld rechts neben dem Lager werden nun geliefert und können ins Lager überführt werden.

3. Unter Berücksichtigung des eigenen Lagerbestandes und der noch nachzuliefernden Fehlmengen ist nun die eigene Bestellung aufzugeben. Die gewünschte Bestellmenge wird auf einem Zettel notiert und verdeckt auf dem Feld *Placed Orders* abgelegt.

4. Jedes Team zeichnet den Lagerbestand und die Bestellmenge der aktuellen Runde sowie die kumulierten Kosten in ein Diagramm ein. Für jeden Spielstein im Lager schlagen 0.50 Geldeinheiten zu Buche, wohingegen jede Einheit, die sich noch auf dem Zettel mit den offenen Bestellungen befindet mit einer Geldeinheit bestraft wird. Die Kosten der aktuellen Runde werden zu den Kosten der bisherigen Runden addiert und im

[13] Vgl. Ossimitz (2002), S.28 f.

Diagramm verzeichnet. Lieferrückstände werden als negativer Lagerbestand abgetragen.

5. Damit ist die Runde vollständig durchgeführt und das Spielbrett kann in die nächste Runde überführt werden. Dazu werden die Einheiten von den rechten *Shipping Delay* Feldern auf die freien linken Felder verschoben sowie die Bestellungen auf den Feldern *Placed Orders* auf die Felder *Incoming Orders* der linken Nachbarn.

Beim Beergame kann den Spielern eine maximale Rundenzahl vorgegeben werden. Bestenfalls jedoch wird das Spiel nach etwa 30-50 Runden durch einen Spielleiter plötzlich abgebrochen. Dadurch wird verhindert, dass die Spieler gegen Ende des Spiels ihre Bestellpolitik ändern und die Lager leer laufen lassen.

2.4 Alternative Variante des Brettspiels

Im Jahr 2002 wurde an der Universität Klagenfurt in Kärnten/Österreich von einer Arbeitsgruppe um Prof. Günther Ossimitz eine veränderte Variante des Beergame-Brettspiels entwickelt. Die Entwicklung verfolge das Ziel, folgende von der Arbeitsgruppe identifizierte Probleme des klassischen Brettspiels zu überwinden[14]:

1. Mangelnder Informationsaustausch innerhalb der Supply-Chain ist einer der Ursachen des Bullwhip-Effekts. Da beim klassischen Brettspiel jedoch alle Spieler auf demselben Spielbrett spielen, können beispielsweise die Lagerbestände von allen Spielern eingesehen werden. Auch Absprachen unter den Spieler sind durch deren räumliche Nähe nur schwer zu unterbinden.

2. Beim klassischen Brettspiel kann es vorkommen, dass sich nicht alle Spieler eines Spiels in der gleichen Runde befinden. Dies ist durch die unabhängige Abarbeitung der Spielschritte durch die Spieler zu erklären. Gehen die Bearbeitungsgeschwindigkeiten der Spieler auseinander, so liegen manchen Spielern zur selben Zeit mehr Informationen vor als anderen.

3. In der klassischen Variante müssen die Spieler den Wert der offenen Bestellungen per Hand berechnen und auf einem Zettel notieren. Durch falsch berechnete Werte werden zuvor bestellte Mengen nicht ordnungsgemäß nachgeliefert, wodurch die Aussagekraft des Spielergebnisses leidet.

[14] Vgl. Ossimitz (2002), S.30, 32

Als Ergebnis wurde eine geänderte Form des Beergame vorgestellt. Anstatt die komplette Supply-Chain auf einem Spielbrett anzuordnen, gibt es bei dieser Variante separate Spielbretter für jede Stufe der Supply-Chain, die auf unterschiedlichen Tischen platziert werden. Dadurch soll der Informationsaustausch zwischen den Spielern unterbunden werden (siehe Abb. 3.2).

Das Spielbrett ist für die gesamte Supply-Chain gleich aufgebaut. Es hat quadratische Form und beinhaltet wiederum fünf quadratische Spielfelder, jeweils eins in jeder Ecke sowie eines in der Mitte. Auf den Feldern in den oberen Ecken werden Bestellungen abgelegt: rechts eingehende Bestellungen (Posteingang), links die eigenen Bestellungen des Spielers (Postausgang). Analog bilden die unteren Felder die Materialwirtschaft ab, links Wareneingang, rechts Warenausgang. Das mittlere Feld ist für offene Bestellungen sowie den Lagerbestand vorgesehen. Wie beim klassischen Brettspiel gibt es auch hier Felder für den Shipping Delay. Diese befinden sich auf einem separaten Spielbrett und Tisch.

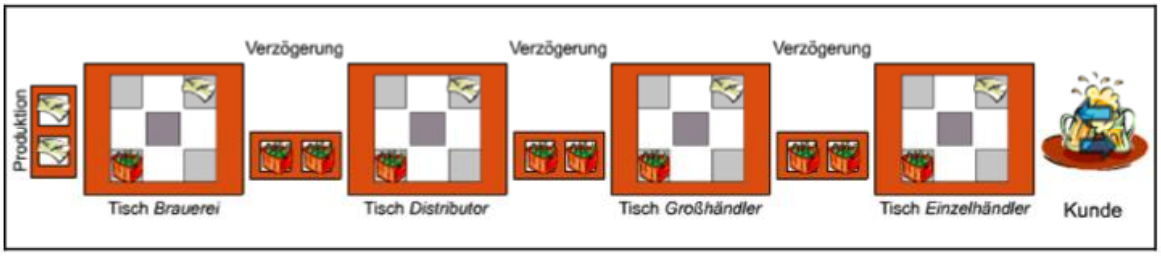


Abb. 2.3: Alternative Variante des Brettspiels[15]

Vor Beginn des Spiels werden die Felder Lager, Shipping Delay, Wareneingang und Posteingang analog zum klassischen Brettspiel derart befüllt, das sich das Spiel in einem ausgeglichenen Startzustand befindet. Eine Spielrunde wird nun nach folgendem Ablauf durchgeführt, der sich von dem der klassischen Variante etwas unterscheidet:

1. Angekommene Waren vom Wareneingang ins Lager überführen.

2. Bestellung vom Feld „Posteingang“ entgegennehmen, zu evtl. bereits vorhandenen offenen Bestellungen addieren und in die Mitte legen.

3. Den kleineren Wert aus Lagerbestand und offenen Bestellungen auf einem Zettel notieren und auf das Feld „Warenausgang“ legen. Werte für Lager und offene Bestellungen aktualisieren.

4. Eigene Bestellung notieren und auf das Feld „Postausgang“ legen.

[15] Ossimitz (2002), S.34

5. Rundenübergang. Zunächst werden wie in der klassischen Variante die Inhalte der *Shipping Delay* Felder vorgerückt. Der Shipping Delay ist mit den Feldern Wareneingang und Warenausgang der umliegenden Supply-Chain-Stufen verbunden. Danach wird die im Feld *Postausgang* liegende Bestellung in den Posteingang des vorgelagerten Spielers gelegt.

Auf das Zeichnen von Diagrammen während des Spiels wird in dieser Variante verzichtet. Stattdessen werden in jeder Runde die Werte für Lagerbestand, eingehende und ausgehende Lieferungen sowie eingehende und ausgehende Bestellungen in eine Tabelle eingetragen. Unter Zuhilfenahme eines Tabellenkalkulationsprogramms können dann die entsprechenden Diagramme aus den Tabellen generiert werden.

Zusätzlich zu den Spielern werden bei dieser Spielvariante zwei weitere Personen pro Spiel benötigt. Zum einen der „Postbote “, der Waren und Bestellungen von Tisch zu Tisch befördert, zum anderen ein „Buchhalter“, der während des Spiels die Rundentabellen in ein Tabellenkalkulationsprogramm übernimmt. Durch den Einsatz des Postboten kann gewährleistet werden, dass sich alle Spieler eines Spiels in derselben Runde befinden, da der Rundenübergang nicht mehr von den Spielern selbst, sondern zeitgleich für alle vom Postboten vollzogen wird. Der Buchhalter überprüft die Werte in den Tabellen der Spieler in der Tabellenkalkulation und kann die Spieler so frühzeitig über eventuell falsche Berechnungen informieren.

2.5 Konsequenzen für eine softwaretechnische Realisierung

Durch die im vorherigen Abschnitt beschriebene Abwandlung des Beergame konnten einige der Probleme des klassischen Brettspiels behoben werden. Allerdings bringt diese Variante wiederum neue Schwierigkeiten mit sich, vor allem in räumlicher und personeller Hinsicht. Wurde beim klassischen Brettspiel pro Spiel nur ein Tisch benötigt, fallen nun ein Tisch pro Supply-Chain-Stufe sowie ein eigener Tisch für den Shipping Delay an. Zudem verlangt die geänderte Variante pro Spiel einen Postboten und einen Buchhalter. Soll eine größere Anzahl an Spielen gleichzeitig ablaufen, so kann dies schnell zu Problemen führen.

Abhilfe kann hier durch den Einsatz von Informationstechnik geschaffen werden. Auf Spielbrett, Spielsteine, Zettel und Kärtchen würde vollständig verzichtet, stattdessen säße jeder Spieler vor einem Computer, der die aktuelle Spielsituation anzeigt und die Bestellung des Spielers entgegennimmt. Sind die Computer der einzelnen Spieler in einem Netzwerk miteinander verbunden, so können die Computer selbst die Weiterleitung aller notwendigen Daten übernehmen. Liegen die Bestellungen aller Spieler vor, kann der Com-

puter das Spiel selbstständig die nächste Runde überführen. Der Rolle des Postboten wäre somit vollständig vom Computer übernommen worden. Auch der Buchhalter entfällt, da sich die Spieldaten bereits auf den Computern befinden. Diese Daten sind, so kein Programmfehler aufgetreten ist, immer in einem konsistenten Zustand, da alle Berechnungen von der Software übernommen werden und die Software ausschließlich gültige Eingaben vom Spieler akzeptiert.

Was die räumlichen Voraussetzungen betrifft, so muss in erster Linie die informationstechnische Infrastruktur vorhanden sein. Geht man von einer Situation im Netzwerk verbundener Computer aus, so ergeben sich prinzipiell drei Möglichkeiten, ein softwarebasiertes Beergame einzusetzen:

1. In einem fest installierten Rechnerverbund, z.B. einem Computerpool.

2. In einem temporären Verbund. Ein solcher ist beispielsweise die Verbindung mehrerer Laptops in einem drahtlosen ad hoc Netzwerk.[16]

3. Eine völlig ortsungebundene Verbindung der Computer über das Internet.

Im weiteren Verlauf soll nun ein detailliertes Konzept zur softwaretechnischen Umsetzung des Beergame erarbeitet und anschließend vollständig implementiert werden.

[16] Vgl. Radtke (2005), S.35 f.

3 Anforderungsanalyse zur softwaretechnischen Realisierung

Zur Ermittlung der Anforderungen an eine softwaretechnische Umsetzung des Beergame können zwei Gruppen von Anforderungen unterschieden werden. Auf der einen Seite der Funktionsumfang, den die Software bereitstellen soll und auf der anderen Seite technische Anforderungen, die sich beispielsweise aus den zur Verfügung stehenden Computer- und Netzwerkinfrastrukturen ergeben. Beide Arten von Anforderungen sollen in diesem Kapitel identifiziert und erläutert werden.

3.1 Ermittlung des Funktionsumfangs

Um bei der Ermittlung des Funktionsumfangs möglichst strukturiert vorzugehen, empfiehlt sich die Zuhilfenahme eines geeigneten standardisierten Vorgehensmodells. Eines der aktuell meistgenutzten Ansätze zur Anforderungsanalyse ist die Beschreibung von Anwendungsfällen in Anwendungsfalldiagrammen. Anwendungsfalldiagramme sind zentraler Bestandteil der Unified Modelling Language (UML) und zielen darauf ab, für jede Art von Benutzer alle möglichen Interaktionen mit der Software zu identifizieren.[17] Zudem können Anwendungsfalldiagramme gut als Grundlage zur Ermittlung eines objektorientierten Datenmodells herangezogen werden.

Ausgangspunkt eines jeden Anwendungsfalldiagramms ist ein *Akteur*. Als Akteur wird in diesem Zusammenhang eine Gruppe von Nutzern gesehen, welche die Software in ähnlicher Weise und mit ähnlichen Intentionen verwendet.[18] Beim Beergame ergeben sich nach dieser Definition zwei Akteure: Der Spieler und der Spielleiter. Während ein Spieler am Beergame teilnimmt, übernimmt der Spielleiter hauptsächlich administrative Tätigkeiten.

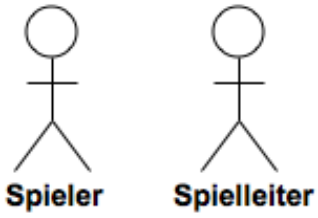


Abb. 3.1: Akteure des Beergame in UML-Notation

Im Folgenden sollen nun für jeden dieser Akteure die dazugehörigen Anwendungsfälle gefunden und im Detail spezifiziert werden.

---

17 Vgl. Jacobson (1992), S.129 f.
18 Vgl. Jacobson (1992), S.127

3.1.1 Anwendungsfälle des Spielleiters

Neben dem Sichtbarmachen des Bullwhip-Effekts ist das Hauptziel des Beergame, den Spielern auch die möglichen Ursachen dieses Phänomens deutlich zu machen, um daraus Gegenmaßnahmen erarbeiten können. Es ist also sinnvoll, das Beergame als praktische Übungseinheit in Lehrveranstaltungen zum Thema Supply-Chain-Management einzubetten. Ein typischer Ablauf einer solchen Übungseinheit sollte sein, das Beergame mehrfach und mit unterschiedlichen Konfigurationen zu spielen: Um den größtmöglichen Bullwhip-Effekt zu bekommen, wird zunächst wird ein Durchlauf mit sehr restriktiven Einstellungen gespielt. Das heißt, den Spielern stehen keinerlei Informationen über die restliche Supply-Chain zur Verfügung. Des Weiteren dürfen sich die Spieler untereinander nicht absprechen. Um den Bullwhip-Effekt zu reduzieren, können nun in weiteren Durchläufen sukzessive die Restriktionen gelöst werden. Der Dozent der Lehrveranstaltung tritt hierbei als Spielleiter des Beergame auf, dem die Aufgabe zufällt, das gewünschte Maß an Restriktionen in der Software einzustellen und einen neuen Spieldurchlauf zu starten. Die Bündelung mehrerer parallel ablaufender Beergame-Spiele mit einer bestimmten Konfiguration soll im Folgenden als Beergame-*Veranstaltung* bezeichnet werden.

Die Anwendungsfälle des Spielleiters können in drei Kategorien unterteilt werden, in solche, die vor Beginn des Spiels, während des Spiels oder nach Beendigung des Spiels durchgeführt werden. Zur Spielvorbereitung muss zunächst eine Veranstaltung angelegt werden. Eine Veranstaltung sollte eindeutig identifizierbar sein, daher sollte zu einer Veranstaltung sowohl einen Titel, als auch eine Kurzbeschreibung erfragt werden. Weiterhin sollten Datum und Uhrzeit der Veranstaltung für spätere Auswertungen gesichert werden. Vom Spielleiter sollten dessen Name sowie seine Email-Adresse erfragt werden. Beim Anlegen der Veranstaltung soll nun durch den Spielleiter die Konfiguration festgelegt werden, die für alle Spiele der Veranstaltung identisch sein wird. Zur Veranstaltungskonfiguration sollte zählen:

- Die Länge der Supply-Chain. Das klassische Brettspiel kann aufgrund des statischen Spielbretts ausschließlich mit vier Stufen gespielt werden. Bei der abgeänderten Variante kann die Supply-Chain durch Einfügen weiterer Spieltische und Shipping Delays verlängert werden. Daher sollte die Länge der Supply-Chain auch bei der Softwarelösung wählbar sein.

- Der Startzustand des Spiels. Beim Brettspiel wurden dazu gewisse Anzahlen an Spielsteinen auf den Spielfeldern platziert sowie Bestellungen auf Zetteln notiert. Um bei einer softwaretechnischen Lösung die höchstmögliche Flexibilität zu erreichen, sollten alle diese Werte durch den Spielleiter vordefinierbar sein. Dazu gehören der anfängliche Lagerbestand und die eingehenden Bestellungen und Lieferungen der ersten Run-

de. Der Shipping Delay sollte sowohl wertmäßig als auch in seiner Länge konfigurierbar sein, um unterschiedliche Vorlaufzeiten abbilden zu können.

- Die Kostenstruktur. Ursprünglich ist vorgesehen, nach jeder Runde für jede Einheit auf Lager 0,5 Geldeinheiten, sowie je eine Geldeinheit für jede Einheit offener Bestellungen zu berechnen. Um eventuelle Zusammenhänge zwischen der Kostenstruktur und dem Spielverhalten untersuchen zu können, sollten die Kostensätze vor Spielbeginn definierbar sein.

- Die Länge des Spiels und der Rundenübergang. Es soll angegeben werden können, nach wie vielen Runden das Spiel beendet werden soll. Zudem kann es sinnvoll sein, dass der Spielleiter für begleitende Erläuterungen die Spiele zeitweise unterbricht, um die ungeteilte Aufmerksamkeit der Spieler auf sich zu ziehen. Daher sollte eine Option vorgesehen werden, bei der ein Spiel erst dann in die nächste Runde überführt wird, wenn dies vom Spielleiter explizit veranlasst wird.

- Die Endkundennachfrage. Beim Brettspiel wurde die Nachfrage der Verbraucher durch einen Kartenstapel repräsentiert, auf dem sich für jede Spielrunde verdeckt eine Karte mit der Höhe der Nachfrage befand. Der Verlauf der Nachfrage ist klassischerweise konstant bis auf einen Sprung in der fünften Runde. Bei der Softwarelösung sollte hier höchstmögliche Flexibilität gewährleistet werden. Es sollte möglich sein, für jede Spielrunde einen eigenen Nachfragewert zu definieren. Auf diese Weise kann sowohl der erwähnte klassische Nachfrageverlauf abgebildet werden, aber auch komplexere Verläufe, welche etwa saisonale Schwankungen berücksichtigen oder speziellen Wahrscheinlichkeitsverteilungen genügen.

- Die Restriktionen. Dazu gehören die Sichtbarkeit von Informationen über die restliche Supply-Chain während des Spiels sowie die Möglichkeit der Spieler, während des Spiels miteinander zu kommunizieren. Als kritisch in Bezug auf die Stärke des Bullwhip-Effekts wurde bereits in Kapitel **Fehler! Verweisquelle konnte nicht gefunden werden.** das Vorhandensein von Informationen über die in Lieferung befindlichen Waren sowie über die aktuelle Endkundennachfrage identifiziert. Es sollte also den Spielern optional erlaubt werden, die Endkundennachfrage sowie den Inhalt der Shipping-Delay-Felder einzusehen. Ursprünglich ist es den Spielern nicht erlaubt, sich untereinander abzusprechen. Es kann jedoch gezeigt werden, dass durch Kommunikation der Bullwhip-Effekt vermindert werden kann. Ist es den Spielern nicht möglich, vor allem beim Spiel über das Internet, sich mündlich abzusprechen, so sollte die Software eine eigene Kommunikationsmöglichkeit bereitstellen, die den Spielern optional zugänglich gemacht werden kann.

Es wird häufig vorkommen, dass ein Spielleiter bestimmte Konfigurationen, die er als angemessen aussagekräftig erachtet, immer wieder mit verschiedenen Spielergruppen verwendet. Um nicht bei jeder Veranstaltung die Konfiguration vollständig neu eingeben zu müssen, sollte eine Möglichkeit geschaffen werden, komplette Konfigurationen speichern zu können, um sie dann bei späteren Veranstaltungen erneut abrufen zu können.

Neben der Festlegung der Spielkonfiguration ist es vor Spielbeginn die Aufgabe des Spielleiters, jeden Spieler einem Spiel und darin einer bestimmten Position innerhalb der Supply-Chain zuzuordnen. Es soll dem Spielleiter die Möglichkeit gegeben werden, Spieler, die sich räumlich recht nahe beieinander befinden, unterschiedlichen Spielen zuzuweisen, um so wiederum Kommunikationsmöglichkeiten zu unterbinden. Spielen räumliche Aspekte eine untergeordnete Rolle, etwa beim Spielen über das Internet, so sollte die Zuordnung der Spieler auch automatisch erfolgen können.

Sind die Spiele gestartet, sollte es dem Spielleiter möglich sein, den Spielverlauf aller laufenden Spiele zu beobachten. Dabei sollten zwei Detaillierungsgrade vorhanden sein. Zum einen eine grobe Übersicht, die für alle Spiele die aktuelle Situation anzeigt, zum anderen eine Ansicht, in der detaillierte Informationen eines einzelnen Spiels abgerufen werden können. Während der Spielleiter in der gröberen Ansicht einen Überblick über den Gesamtfortschritt der Veranstaltung erhält, soll in der Detailansicht der vollständige Spielverlauf sichtbar sein. Über eine zusätzliche Ausgabe der Daten in Diagrammform kann der Spielleiter die Ausprägung des Bullwhip-Effekts in jedem Spiel überprüfen.

Während des Spielbetriebs kann es die Situation erfordern, einige der vor dem Spiel gesetzten Einstellungen anpassen zu müssen. So kann es z. B. passieren, dass die gewählte maximale Rundenzahl nicht in der zur Verfügung stehenden Zeit durchzuführen ist. Hier muss dem Spielleiter die Möglichkeit gegeben werden, die maximale Rundenzahl bei laufendem Spiel anpassen zu können. Ähnliche Probleme können auch die Endkundennachfrage betreffen. Stellt sich während des Spiels heraus, dass der gewählte Nachfrageverlauf nur zu einem unzureichenden Bullwhip-Effekt führt, so sollte der Spielleiter die Nachfrage der kommenden Runden anpassen können.

Während Kommunikation zwischen den Spielern zumeist nicht erwünscht ist, steht dennoch der Spielleiter in ständigem Kontakt zu den Spielern. Zum einen kann er den Spielern theoretische Hintergründe zum Thema vermitteln, zum anderen kann sich die Kommunikation auch um die Durchführung des Spiels drehen. Da der Spielleiter stets Einsicht in den aktuellen Verlauf der Spiele hat, kann er jedem Spieler gezielt Feedback zu seiner Spielweise geben. Da Spieler und Spielleiter, wie bereits erwähnt nicht zwingend am selben Ort sein müssen, sollte auch für diesen Zweck ein zusätzlicher Kommunikationskanal vorhanden sein.

Nach Beendigung der Spiele sollten dem Spielleiter sämtliche während der Veranstaltung angefallenen Daten dauerhaft zur Verfügung stehen. Diese können dann nach der Veranstaltung aufbereitet und den Spielern vorgelegt werden, mit dem Ziel, Ursachen und Wirkung des Bullwhip-Effekts zu analysieren und Gegenmaßnahmen zu erarbeiten. Zur Vereinfachung der Datenaufbereitung ist es sinnvoll, die Daten in einer für die Verwaltung und Aufbereitung numerischer Daten geeigneten Form ausgeben lassen zu können, die beispielsweise von einem Tabellenkalkulationsprogramm gelesen werden kann.

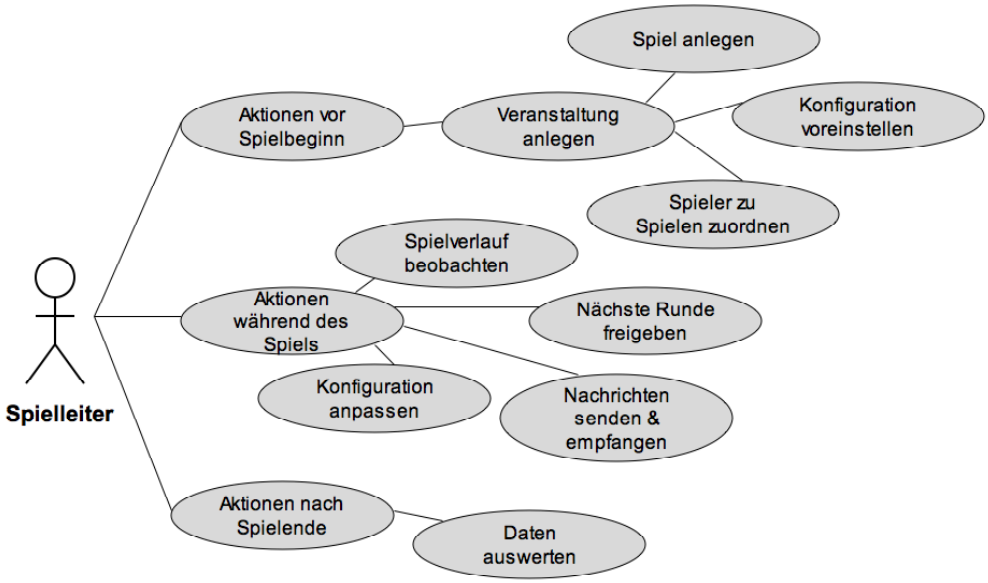


Abb. 3.2: Anwendungsfalldiagramm des Spielleiters

3.1.2 Anwendungsfälle des Spielers

Nachdem sämtliche administrativen Funktionen entweder vom Spielleiter oder von der Software übernommen werden, bleibt die Hauptaufgabe des Spielers die Verwaltung seines Lagerbestands.

Zunächst jedoch muss sich jeder Spieler bei der vom Spielleiter erstellten Veranstaltung anmelden. Dabei sollten Daten vom Spieler erfragt werden, die den Spieler identifizieren und einen späteren Kontakt möglich machen. Über den Kontakt kann der Spielleiter dem Spieler evtl. nachträglich eine Nachbereitung der Veranstaltung zukommen lassen. Die Abfrage des Namens und einer gültigen Email-Adresse des Spielers erscheint hier ausreichend.

Nachdem das Spiel vom Spielleiter gestartet wurde, muss es dem Spieler möglich sein, sich die aktuelle Datenlage des Spiels, sowie auch den Spielverlauf anzusehen. Es soll visualisiert werden, in welcher Weise die einzelnen Daten des Spiels miteinander in Beziehung stehen, also wie sie berechnet werden. Hier bietet sich an, zwei alternative Spielan-

sichten zu realisieren, zwischen denen der Spieler frei wechseln kann: Zum einen eine tabellarische Ansicht aller Daten der bisher gespielten Runden, zum anderen eine Ansicht, welche nur die Daten der aktuellen Runde enthält und Transformation der Daten bei jedem Rundenübergang in einer Animation verdeutlicht. Die Aktualisierung der Benutzeroberfläche soll, wie beim Spielleiter, automatisch und in annähernder Echtzeit, also ohne größeren Zeitverzug geschehen.

Außerdem soll angedeutet werden, welche Daten wegen unzureichendem Supply-Chain-Management nicht zur Verfügung stehen, beispielsweise die in den nächsten Runden ankommenden Liefermengen. Dadurch soll dem Spieler ein Hinweis auf eine mögliche Ursache des Bullwhip-Effekts gegeben werden.

Der Spieler nimmt die ihm angezeigten Daten als Grundlage für seine Bestellentscheidung in der jeweiligen Runde. Die Bestellung neuer Ware ist im Grunde die einzige verbliebene Spielaktion des Spielers. Da irrtümliche Fehleingaben nie ausgeschlossen werden können, sollte es dem Spieler möglich sein, seine Bestellung zu noch korrigieren, solange die nächste Runde noch nicht freigegeben wurde.

Durch den bereits beim Spielleiter eingeführten Kommunikationsdienst kann der Spieler Nachrichten vom Spielleiter empfangen. Da unzureichende Kommunikation innerhalb der Supply-Chain einer der Gründe für den Bullwhip-Effekt ist, sollte es den Spielern zudem erlaubt werden können, den Kommunikationsdienst auch untereinander zu nutzen. Auf diese Weise können die Spieler den Bullwhip-Effekt reduzieren, indem sie diesen Kommunikationskanal zur Angleichung des Informationsstandes über die gesamte Supply-Chain hinweg nutzen.

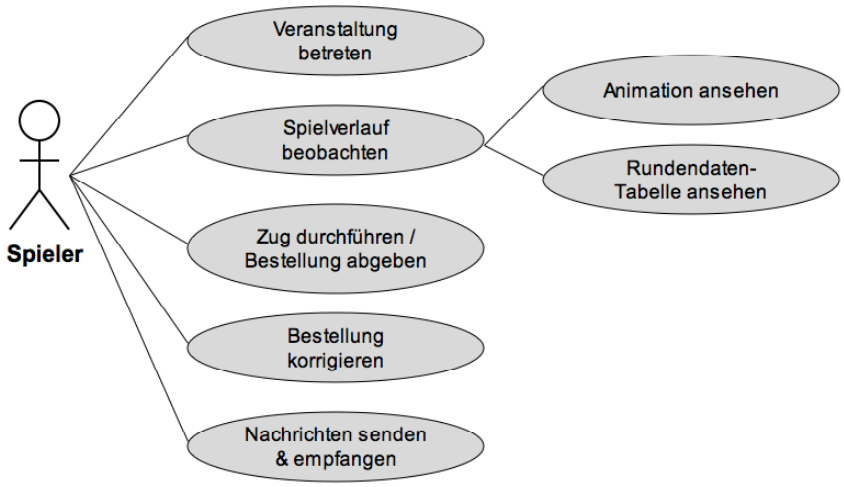


Abb. 3.3: Anwendungsfalldiagramm des Spielers

3.2 Technische Anforderungen

Damit bei der Nutzung der Software keinerlei Nutzeneinbußen gegenüber dem Brettspiel zu verzeichnen sind, sind folgende zusätzliche Anforderungen zu berücksichtigen, welche nicht zum direkten Funktionsumfang der Software zu zählen sind, sondern vielmehr auf den Einsatz der Informationstechnik zurückzuführen sind:

- Der Aufwand zur technischen Vorbereitung einer Beergame-Veranstaltung sollte unter allen Umständen minimiert werden. Bestenfalls sind auf den Computern der Spieler keinerlei Installationen nötig. Auf dem Computer des Spielleiters sollte nach einer einmaligen Vorkonfiguration die Software dauerhaft nutzbar sein.

- Durch den Einsatz der Software sollten keine laufenden Lizenzkosten entstehen, d. h., es sollten nur solche externe Komponenten verwendet werden, die sowohl für private, als auch für kommerzielle Zwecke kostenlos nutzbar sind.

- Die Benutzerschnittstelle der Software sollte derart angelegt sein, dass sämtliche darin enthaltenen Textelemente in ihrer Sprache wählbar sind. Die Auswahl der Sprache obliegt dem Benutzer. Die Möglichkeit des späteren Hinzufügens weiterer Sprachen sollte vorgesehen werden.

- Beim Einsatz von Informationstechnik ist das Auftreten von Fehlern nie auszuschließen. Mögliche Fehlerquellen sind neben der Software selbst z. B. das Betriebssystem oder die Netzwerkverbindung. Damit durch einen Fehler nicht ein ganzes Spiel verloren geht, sollte die Möglichkeit geschaffen werden, nach einem Fehler ein Spiel in einem möglichst aktuellen Zustand wieder aufnehmen zu können.

4 Objektorientierte Modellierung des Datenmodells

Bereits im vorherigen Kapitel wurden die Anwendungsfalldiagramme mit der Begründung eingesetzt, dass sie sich gut als Grundlage zur Erarbeitung eines objektorientierten Datenmodells eignen. Die Verwendung objektorientierter Programmiertechniken ist keine explizite Anforderung an die zu entwickelnde Software, allerdings ist die Objektorientierung Grundlage fast aller moderneren Programmiersprachen und kann somit als geltender Standard angesehen werden.

Ziel dieses Kapitels ist die Identifizierung von Objektklassen im Beergame sowie deren Beziehungen untereinander. Zusammen mit elementaren Attributen und Methoden der Klassen soll ein erstes Klassendiagramm[19] erstellt werden, das als Grundlage der späteren Implementierung dienen soll. Dieses Klassendiagramm ist noch unabhängig von einer Programmiersprache, es werden ausschließlich elementare oder abstrahierte Datentypen verwendet. In der Regel werden in der Implementierungsphase die Klassen dann um weitere Hilfsmethoden erweitert werden, zudem werden Hilfsklassen hinzukommen, die jedoch unabhängig vom Datenmodell des Beergame sind.

Ein üblicher Weg zur Identifizierung der Klassen besteht darin, die Beschreibungen der Anwendungsfälle nach Begriffen zu durchsuchen, die im vorliegenden Modellierungskontext untereinander in Verbindung stehen und denen Attribute oder Operationen zugeordnet werden können.[20] Wendet man diese Methode auf die in Kapitel 3.1 gefundenen Anwendungsfälle an, so können die folgenden Begriffe extrahiert werden:

- Veranstaltung
- Spiel
- Spielrunde
- Spieler
- Spielleiter
- Veranstaltungskonfiguration
- Spielverlauf
- Bestellung
- Nachricht

---

19 Vgl. Jacobsen, Booch, Rumbaugh (1999), S. 218 ff.

20 Vgl. Balzert (1996), S.360

Im Folgenden soll nun zunächst für jeden dieser Begriffe überprüft werden, ob die Schaffung einer eigenen Klasse angemessen erscheint. Ist dies der Fall, sollen die dazugehörigen Attribute und Methoden gefunden werden, insofern sie bereits aus der Anforderungsanalyse hervorgehen.[21] Da zur internationalen Lesbarkeit des Quellcodes Klassennamen stets mit englischen Bezeichnungen versehen werden, soll dieser Umstand nachfolgend zwar bereits berücksichtigt, aber erst in der Implementierungsphase endgültig angewendet werden.

Veranstaltung (Event)

Eine Veranstaltung enthält einen *Spielleiter*, mehrere *Spiele* sowie eine *Veranstaltungskonfiguration*. Weitere Attribute sind ein Titel und eine Beschreibung in Form von Zeichenketten sowie das Datum der Veranstaltung. Zur Speicherung des Datums sollten universelle Zeitstempel verwendet werden, um von der unterliegenden Programmiersprache unabhängig zu sein. Diese Zeitstempel sind ganzzahlige numerische Werte.[22] Die Zuordnung von Spielern zu Spielen soll von der Klasse *Veranstaltung* übernommen werden. Noch nicht zugeordnete Spieler werden daher zunächst von der Veranstaltung aufgenommen (im Attribut *pendingPlayers*), bevor sie anschließend an ein Spiel weitergegeben werden. Eine Methode zur automatischen Zuordnung soll außerdem in der Klasse *Veranstaltung* verortet sein.

Spiel (Game)

Ein Spiel gehört zu genau einer *Veranstaltung*. Es kann mehrere *Spieler* aufnehmen, die genaue Anzahl richtet sich nach der *Veranstaltungskonfiguration*. Ein Spiel hat immer einen bestimmten Status: es ist entweder noch nicht gestartet, bereits beendet, oder es läuft noch. In letzterem Fall muss die Rundenzahl gespeichert werden, in der sich das Spiel gerade befindet. Der Spielstatus kann durch eine eindeutig vergebene Identifikationsnummer abgebildet werden. Die Klasse *Spiel* koordiniert den Spielfluss, indem Methoden bereitgestellt werden, die das Spiel anfänglich initialisieren und danach bei jedem Rundenübergang Lieferungen und Bestellungen zwischen den Spielern transferieren.

Spieler (Player)

Ein Spieler wird durch seinen Namen und seine Email-Adresse identifiziert, die jeweils durch eine Zeichenkette abgebildet werden können. Ein Spieler gehört immer zu genau einem Spiel und absolviert darin mehrere *Spielrunden*. Pro Runde gibt der Spieler eine

[21] Auf die Angabe trivialer Methoden zum Setzen und Auslesen der Attribute soll hier verzichtet werden.
[22] technische Details folgen in Kap. 6.1.2

Bestellung ab. Bei jedem Rundenübergang bekommt die Klasse *Spieler* von der Klasse *Spiel* die aktuelle Lieferung sowie die aktuelle Bestellung übergeben. Die Klasse *Spieler* schließt damit die aktuelle Runde ab und berechnet die Werte der nächsten Runde.

Spielleiter (Admin)

Analog zum Spieler werden auch vom Spieleiter Name und Email-Adresse aufgenommen. Ein Spielleiter ist mit genau einer Veranstaltung verknüpft. Nur ihm ist es gestattet, die Methoden der Veranstaltung auszuführen. Zudem kann er Nachrichten an Spieler der Veranstaltung senden sowie Nachrichten von Selbigen empfangen.

Veranstaltungskonfiguration (EventPreferences)

Die Konfiguration nimmt sämtliche Einstellungen auf, die der *Spielleiter* vor Spielstart tätigt, bzw. während des Spiels anpasst. Nach der Anforderungsanalyse sind diese:

- Die Benennung der Stufen der Supply-Chain durch je einen Wert vom Typ String in einer Liste.
- Der Verlauf der Endkundennachfrage. Da diese oft bis auf wenige Sprünge konstant ist, bietet sich an, nur die Änderungen im Nachfrageverlauf zu speichern. Die Nachfrage wird daher durch eine Menge von Tupeln der Form {Runde, Änderung} spezifiziert. Die Menge muss zur Initialisierung stets ein Wertepaar mit dem Rundenwert 1 beinhalten.
- Die maximale Rundenzahl als numerischer Wert. Die Art der Rundenfreigabe (automatisch oder manuell) kann analog zum Spielstatus durch eine eindeutige Nummer identifiziert werden.
- Länge und anfängliche Befüllung des Shipping Delays. Hierzu kann eine Liste dynamischer Länge genutzt werden, deren Inhalt je ein numerischer Wert pro Eintrag ist.
- Anfängliche Befüllungen des Lagerbestands sowie der eingehenden Bestellung und Lieferung durch je einen ganzzahligen numerischen Wert.
- Die Kostensätze für Lager- und Fehlmengen. Hier sind numerische Werte mit Dezimalstellen erlaubt.
- Sichtbarkeit von Endkundenachfrage und Shipping Delay für Spieler sowie die Verfügbarkeit der Kommunikationsfunktion durch boolesche Ausdrücke.

Da Veranstaltungskonfigurationen dauerhaft gespeichert werden sollen, um sie in weiteren Veranstaltungen wieder zu verwenden, werden zu jeder Konfiguration ein Name sowie eine Kurzbeschreibung als String aufgenommen.

Spielrunde (RoundData)

Eine Spielrunde muss alle Daten aufnehmen, die in einer Runde bei einem *Spieler* anfallen. Diese sind:

- der Lagerbestand zu Beginn der Runde,
- offene Bestellungen zu Beginn der Runde,
- der Inhalt des Shipping Delays,
- die eingehende Lieferung,
- die eingehende Bestellung,
- die ausgehende Lieferung,
- der neue Lagerbestand,
- der neue Auftragsbestand,
- die kumulierten Gesamtkosten, sowie
- die eigene (ausgehende) Bestellung.

Bestellung

Der Bestellwert ist bereits eines der Attribute der Spielrunde. Zudem sind sämtliche Methoden zur Handhabung von Bestellungen letztlich Aktionen des Spielers und werden daher dessen Klasse zugeordnet.

Spielverlauf

Der Spielverlauf kann durch eine Liste von Spielrunden abgebildet werden. Diese kann direkt als Attribut des Spielers umgesetzt werden. Zudem können dem Spielverlauf keine eigenen Methoden zugewiesen werden, da die Klasse *Spieler* den Rundenübergang koordiniert. Somit wird für den Spielverlauf keine eigene Klasse benötigt.

Nachricht (Message)

Eine Nachricht hat typischerweise einen Sender und einen oder mehrere Empfänger. Sowohl beim Sender als auch bei den Empfängern kann es sich um *Spieler* oder *Spielleiter* handeln. Weiterhin enthält eine Nachricht einen Nachrichteninhalt, für den in diesem Fall eine rein textliche Darstellung ausreichend ist. Auf eine sonst übliche Betreffzeile wird in diesem Kontext verzichtet, um den Vorgang des Nachrichtenschreibens im oft etwas hektischen Spielbetrieb zu beschleunigen.

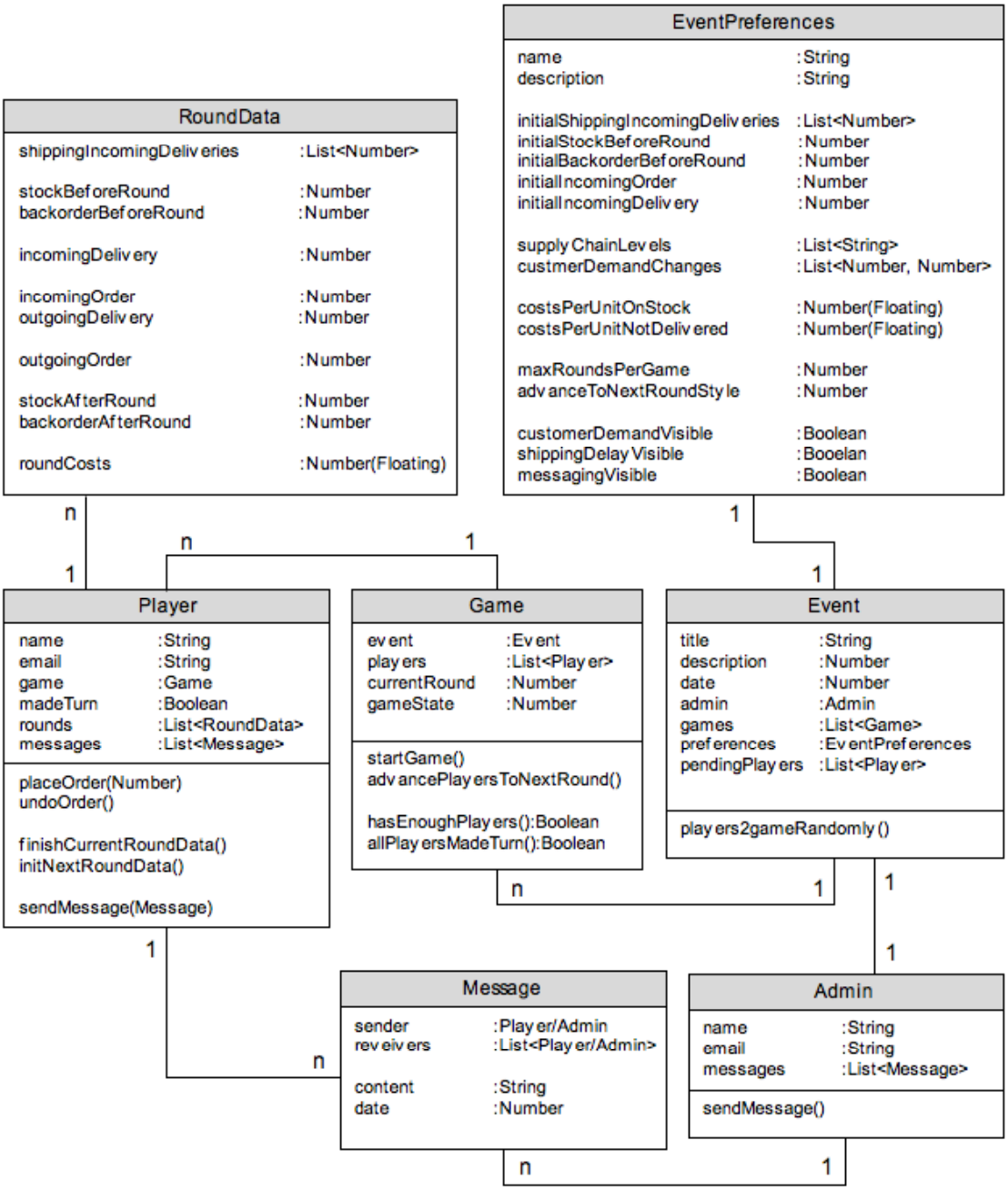


Abb. 4.1: Klassendiagramm vor der Implementierungsphase

5 Softwarearchitektur

5.1 Erarbeitung der Softwarearchitektur

Nachdem in Kapitel 4 die Datenstruktur des Beergame definiert wurde, ist es nun Ziel dieses Abschnitts, eine Softwarearchitektur zu erarbeiten. Als Softwarearchitektur wird hier die „strukturierte oder hierarchische Anordnung der Systemkomponenten sowie die Beschreibung ihrer Beziehungen“ verstanden.[23] Bei der Erstellung der Architektur wird der Fokus vor allem darauf zu legen sein, die spätere Einsatzsituation der Software bestmöglich abzubilden. Damit ist hauptsächlich der Zugriff der Benutzer auf das Beergame gemeint sowie die Frage, auf welcher Maschine sich die Programmlogik und die anfallenden Daten der Software befinden werden. Nach der Festlegung der Architektur wird schließlich herausgearbeitet, welche Techniken, d. h. welche Programmiersprachen, Laufzeitumgebungen und Datenformate bei der Umsetzung der Anwendung, sowie für Datenaustausch und Datenhaltung zum Einsatz kommen sollen.

In Kapitel 3 wurde herausgearbeitet, dass es hauptsächlich zwei Einsatzsituationen für das Beergame gibt: Eine Seminarsituation, bei welcher der Zugriff auf die Software über ein lokales Netzwerk erfolgt, oder der entfernte Zugriff über das Internet. Der Zugang zum lokalen Netzwerk kann entweder über die persönlichen Laptops der Teilnehmer oder über fest installierte Rechner eines Computerpools erfolgen. Fasst man diese Anforderungen zusammen, so stellen sich zwei alternative Ansätze zur Realisierung der Software heraus:

1. Eine Desktopapplikation, die auf den Rechnern der Spieler und des Spielleiters installiert wird. Die Anwendungen kommunizieren über die explizite Implementierung der Netzwerkfunktionalitäten peer-to-peer[24] miteinander.

2. Eine Browseranwendung, die auf einem Webserver betrieben wird und von Spielern im Netzwerk über eine Eingabe einer URL-Adresse in einem Browser zu erreichen ist.

Es ist ein Vorteil der Browserlösung, dass auf den Rechnern der Spieler keinerlei Installationen vorgenommen werden müssen, vorausgesetzt es ist ein Browser vorhanden. Dies kann die Vorbereitungszeit auf eine Veranstaltung erheblich reduzieren, vor allem wenn das Beergame nicht auf stationären Computern, sondern auf den Notebooks der Teilnehmer ablaufen soll. Des Weiteren muss die Netzwerkkommunikation zwischen den Spielern nicht explizit konstruiert und implementiert werden, da bei dieser Lösung die Daten aller Spieler auf einem Webserver zusammenkommen.

---

23 Balzert (1996), S.639

24 Vgl. Peterson, Davie (2003), S.690 ff.

Für die Entwicklung einer Desktopanwendung spricht dagegen die Tatsache, dass bei einer solchen Lösung eine Trennung von Programm- und Präsentationslogik nicht nötig ist. Der Wegfall der Schnittstellen zwischen Browser und Webserver reduziert im Allgemeinen die Komplexität des Quellcodes. Diese Einsparungen werden allerdings durch die komplexere Handhabung der Netzwerkkommunikation zum Teil wieder aufgehoben.

Die geforderte „Echtzeitfähigkeit“ des Beergame, also der regelmäßige Abgleich der Daten aller Spieler und die automatische Aktualisierung der Benutzeroberfläche sind sowohl mit einer Desktop-, als auch mit einer Browseranwendung zu bewerkstelligen. Dabei ist zu beachten, dass bei einer Browseranwendung die Auslastung des Webservers mit der Anzahl der gleichzeitig verbundenen Spieler ansteigt. Es ist jedoch nicht davon auszugehen, dass durch die übliche Anzahl der Spieler eine Überlastung des Servers verursacht werden könnte, da auch Aktualisierungsintervalle von etwa 2 bis 5 Sekunden für diesen Fall durchaus akzeptabel sind.

Somit fällt die Wahl auf die Entwicklung einer Browseranwendung, womit wir es mit einer klassischen Client/Server – Architektur zu tun haben werden.[25] Bei einer solchen werden die Programmlogik und die Datenhaltung vom Server übernommen, wohingegen die Clients nur die Anzeige der vom Server zur Verfügung gestellten Daten sowie das Entgegennehmen von Benutzereingaben übernehmen. Server und Clients kommunizieren über das Netzwerk, wobei es für die Entwicklung der Anwendung keinen Unterschied ergibt, ob sich Clients und Server im selben lokalen Netz befinden, oder über das Internet verbunden sind. Die nachfolgende Abbildung zeigt eine erste Zusammenfassung der Architektur, welche im Folgenden noch vervollständigt werden wird.

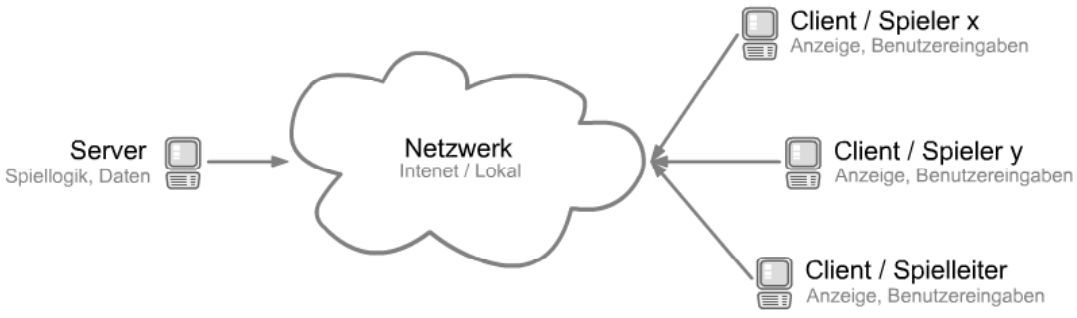


Abb. 5.1: Client/Server Architektur der Beergame-Software

[25] Vgl. Tanenbaum, Stehen (2003), S.61 ff.

5.2 Auswahl der Techniken zur Umsetzung der Architektur

Bei der Entscheidung, auf welche Techniken die Umsetzung des Beergame fußen soll, erscheint es am sinnvollsten, zunächst der Frage nachzugehen, auf welche Weise das Frontend umgesetzt werden soll. Unter dem Begriff „Frontend“ ist hier der Teil der Software gemeint, der auf den Clients ausgeführt wird, also hauptsächlich die Umsetzung der Benutzerschnittstelle. Erst danach ist es möglich, adäquate Aussagen darüber zu treffen, in welcher Form der Server Daten und Anfragen entgegennimmt und beantwortet.

5.2.1 Technologien für das Frontend

Wie bereits beschrieben wurde, soll das Beergame als Webanwendung implementiert werden, d.h. es soll den Benutzern über einen Browser zugänglich sein. Daten und Programmlogik liegen auf einem Webserver.

Zur Umsetzung eines solchen Ansatzes kommen grundsätzlich drei Techniken in Frage, die im Folgenden näher untersucht und evaluiert werden sollen. Diese sind:

1. die Verwendung dynamisch generierter Webseiten über serverseitige Scriptsprachen wie PHP oder Java Server Pages (JSP), oder über Java Servlets,

2. die Verwendung von HTML zusammen mit clientseitigem JavaScript zur Einbindung dynamischer Inhalte, auch bekannt als AJAX -Webseiten, oder

3. die Verwendung von clientseitigen Plugins, welche im Browser ablaufen, und ihre Daten von Servern beziehen können, wie z. B. Java Applets oder Adobe-Flash-Filme.

5.2.1.1 Serverseitiges Skript und Servlets

Dynamische Webseiten funktionieren nach folgendem Prinzip: Der Benutzer stellt über einen Link oder die Eingabe einer Adresse eine Anfrage an den Webserver. Der Server wertet diese Anfrage aus und stellt die angeforderten Daten zusammen. Diese Daten werden ins HTML-Format konvertiert und zusammen mit weiteren HTML-Bausteinen zu einer Webseite zusammengefügt. Dem Benutzer bleibt die „Dynamik“ der Seite verborgen, denn letztlich kommt im Browser nur das fertige HTML-Dokument an, so wie es auch bei statischen Seiten der Fall ist.

Eine Möglichkeit zur Umsetzung solcher dynamischer Seiten ist die Verwendung von serverseitig ausgeführten Scriptsprachen. Zu diesen zählen etwa die gängigen Sprachen PHP und JSP (Java Server Pages). Bei diesen bildet ein HTML-Dokument die Grundlage, in das

Skriptelemente integriert werden können. Dazu gehören typische Skriptelemente wie Schleifen und Verzweigungen, aber auch Funktionen zur Anbindung von Datenbanken und XML-Dokumenten. In JSP können prinzipiell alle existierenden Javabibliotheken angesprochen werden, daher sind tendenziell bei komplexeren Projekten JSP gegenüber PHP zu bevorzugen. Diese Skriptelemente werden dazu genutzt, die HTML-Seite mit genau dem Inhalt zu befüllen, wie er in den Aufrufparametern verlangt wurde.[26] Bei einem Aufruf der Seite werden die der Adresse angefügten Parameter vom Server ausgelesen und die im Dokument enthaltenen Skriptbausteine ausgeführt. Ergebnis ist ein konventionelles HTML-Dokument, das der Server zum Browser des Benutzers zurücksendet.

Bei Java Servlets wird in gewisser Weise ein umgekehrter Weg beschritten: Ein Servlet ist eine Java-Klasse, die, genau wie eine HTML-Seite, über einen Webserver per Internetadresse erreichbar ist. Bei einer Anfrage wird eine bestimmte Methode des Servlets aufgerufen und die Parameter der Anfrage übergeben. In dieser Methode kann nun der HTML-Code generiert, und als Rückgabewert der Methode zum Browser zurückgeschickt werden. Anstelle einer HTML-Seite, die an bestimmten Stellen mit Programmcode versehen wird, dient bei Servlets eine Java-Klasse als Grundlage.

Technisch gesehen sind sowohl mit JSP als auch mit Servlets dieselben Funktionen umsetzbar, da der Webserver eine JSP-Seite bei jeder Anfrage zunächst in ein Servlet übersetzt.[27] Die Entscheidung zwischen JSP und Servlets wird daher meist anhand des dynamischen Anteils am Inhalt der HTML-Seiten getroffen.

Sowohl mit serverseitigem Skript als auch mithilfe von Servlets ist es möglich, die beim Beergame anfallenden Daten aufzubereiten und den Benutzern im Browser darzustellen. Auch die Eingaben der Benutzer können über HTML mit Verweisen und Formularen realisiert werden. Leider kann jedoch die für das Beergame geforderte Aktualisierung des Bildschirms in angenäherter Echtzeit nur unzureichend umgesetzt werden: Wie bereits erläutert, wird bei jeder Anfrage an den Server die HTML-Seite neu aufgebaut und an den Browser geschickt. Zur Umsetzung der Echtzeitaktualisierung müsste diese Anfrage im Abstand weniger Sekunden kontinuierlich wiederholt werden. Als Konsequenz wäre ein ständiges Neuladen der kompletten Seite auf dem Bildschirm zu sehen, was offensichtlich erstens durch die unvorteilhafte optische Anmutung sowie zweitens durch ein ständiges Verwerfen der vom Benutzer getätigten Eingaben nicht akzeptabel ist.

[26] Vgl. Bergsten (2002), S.25 f.
[27] Vgl. Bergsten (2002), S.6

5.2.1.2 Rich Internet Applications

Die Lösung des Problems des vollständigen Neuladens von Webseiten ist als eines der wichtigsten Errungenschaften des Konzepts der Rich Internet Applications, RIA abgekürzt, anzusehen.[28]

Der Begriff „Rich Internet Application“ geht auf das Jahr 2002 zurück und stammt ursprünglich aus einem Whitepaper der Firma Macromedia über die Veröffentlichung des neuesten Flash-Players.[29] Der Begriff stellt eine Verbindung aus den Bezeichnungen „Rich Client“ und „Internet Application“ dar. Ein Rich Client bezeichnet eine Anwendung mit komplexer Benutzerschnittstelle. Zur Interaktion zwischen Benutzer und Software steht eine Reihe von Steuerelementen zur Verfügung, die über einfache Formularelemente wie Schaltflächen und Eingabefelder hinausgehen. Zu diesen zählen beispielsweise verschachtelte Menüstrukturen, Drag&Drop-Funktionalitäten, baumartig strukturierte Listen oder voneinander abhängige Dialogfenster. Dabei ist eine wesentliche Eigenschaft von Rich Clients, dass sich die Anzeige selbstständig und unverzüglich an genau den Stellen aktualisiert, an denen sich die dahinter liegenden Daten geändert haben. Hauptsächlich findet man Anwendungen mit Rich-Client-Eigenschaften als installierte Desktopanwendungen auf PCs. In letzter Zeit wird jedoch vermehrt daran gearbeitet, Rich-Client-Funktionalitäten auch auf anderen Plattformen wie z. B. Handhelds, Mobiltelefonen oder, wie im Folgenden dargelegt wird, auch innerhalb eines Browsers lauffähig zu machen.

Dem gegenüber steht die „Internet Application“, die im bisherigen Verlauf dieser Arbeit als „Webanwendung“ bezeichnet und bereits beschrieben wurde. Diese sollen im weiteren Verlauf als „klassische Webanwendungen“ bezeichnet werden, um sie stärker von den hier eingeführten Rich Internet Applications abzugrenzen. Klassische Webanwendungen unterscheiden sich in Aussehen und Benutzerführung stark von Desktopanwendungen. Das Problem des vollständigen Neuladens wurde bereits erwähnt, dazu kommt, dass die Anzeige weniger die Erscheinung von Eingabemasten und Dialogen besitzt, sondern mehr das Aussehen von formatierten Textseiten aufweist. Zudem stellt das für Webanwendungen benutzte HTML viele aus Desktopanwendungen bekannte Funktionen wie Menüleisten oder auch Drag&Drop originär nicht zur Verfügung.

Rich Internet Applications verbinden die beiden genannten Ansätze. Um ein vollständiges Neuladen der Seite zu verhindern, wird die Verbindung zum Server ausschließlich zum Datenaustausch verwendet. Die Einbindung der empfangenen Daten in die Seite geschieht nicht mehr, wie bei klassischen Webanwendungen, auf dem Server, sondern direkt im

---

28 Vgl. Vossen, Hagemann (2007), S.136.
29 Vgl. Allaire (2002), S.2

Browser des Benutzers. Neben der Unterbindung des Neuladens ergibt sich noch ein weiterer Vorteil dieses Ansatzes: Die zur Verfügung stehende Bandbreite der Netzwerkverbindung wird effizienter genutzt, da die Benutzeroberfläche nur noch beim ersten Aufruf der Seite geladen werden muss und danach ausschließlich Daten ausgetauscht werden. Zudem wird versucht, über einen sehr großen Einsatz von browserseitig ablaufenden Skripten, wie z. B. JavaScript oder Flash, die oben genannten Eigenschaften komplexer Benutzeroberflächen nachzuahmen.

Einer der Ziele bei der Entwicklung von Rich Internet Applications ist es, die Funktionalitäten von Desktopapplikationen über das Internet verfügbar zu machen, und über diesen Weg die Benutzer unabhängig von einem bestimmten Computer zu machen. So existieren beispielsweise Implementierungen von Email-Programmen als RIA, die sich in der Art der Bedienung von ihren lokal installierten Gegenstücken nur wenig unterscheiden.[30] Die Daten werden bei solchen Anwendungen meist nicht lokal, sondern auf einem Server gehalten, sodass der Benutzer unabhängig von seinem Standort nur einen Browser benötigt, um auf Programme und Daten zugreifen zu können. Es wird in diesem Zusammenhang auch von einer kontinuierlichen Verlagerung des Desktops ins Internet gesprochen.[31]

Einige Anbieter von Entwicklungsframeworks für Rich Internet Applications geben sogar an, dass RIA, die auf diesen Frameworks basieren, ohne Anpassungen sowohl in Browsern als auch auf Mobiltelefonen und Handhelds angezeigt werden können.[32] Damit wird nicht nur die Unabhängigkeit von einem bestimmten Computer angestrebt, sondern in Zukunft auch vermehrt die Unabhängigkeit von einem bestimmten Gerätetyp.

Gleichwohl müssen auch kritische Punkte zum RIA-Konzept erwähnt werden. Zum einen darf nicht angenommen werden, dass man mit einer RIA die Benutzerfreundlichkeit oder die Leistungsfähigkeit von Desktopanwendungen vollständig erreichen kann. Zwar können die grafischen Steuerelemente eines Betriebssystems mittlerweile hinreichend genau approximiert werden, jedoch zeigen sich die Grenzen z.B. im Zusammenspiel von RIA und Desktopanwendungen etwa über Drag&Drop. Des Weiteren stellt die Tatsache, dass RIA im Browser ausgeführt werden, gewissermaßen eine natürliche Grenze dieses Konzepts dar. Aus dem Browser heraus ist es weder möglich, uneingeschränkt auf das lokale Dateisystem zuzugreifen, noch können externe Programme aufgerufen werden. Dies sind offensichtlich für einen Browser essenzielle Sicherheitsmechanismen, jedoch schränken sie den möglichen Funktionsumfang einer RIA erheblich ein.

---

30 z.B. Zimbra (www.zimbra.com) oder LaszloMail (www.laszlomail.com)

31 Vgl. O´Reilly (2005)

32 Beispielsweise mit dem *OpenLaszlo* Framework, www.openlaszlo.org

In den nächsten Abschnitten werden nun die zwei gängigsten Ansätze zur Realisierung von Rich Internet Applications vorgestellt sowie auf ihre Eignung für die durchzuführende Implementierung des Beergame untersucht. Diese sind zunächst der Ajax-Ansatz und im Anschluss die Verwendung von Browser-Plugins.

Die AJAX-Technologie

Im Bereich der Softwareentwicklung ist der Begriff AJAX ein Akronym, das für „Asynchronous Javascript and XML“ steht. Ajax ist weniger eine Technologie, denn eine Kombination bereits vorhandener Ansätze zur Entwicklung von Webseiten, die vor allem bei Internetanwendungen des sog. „Web 2.0“ häufig genutzt wird.[33] Ajax verbindet folgende Technologien:

1. *DHTML*, welches für die Anzeige im Browser verwendet wird. Das von Ajax genutzte DHTML unterscheidet sich von normalem HTML durch die Eigenschaft, auch nach dem Laden per Skript in seiner Struktur manipulierbar zu sein.

2. *JavaScript*. Nachdem die Seite geladen wurde, wird jegliche nachträgliche Veränderung über Javascript durchgeführt. Javascript beinhaltet dazu eine Schnittstelle, über die das HTML-Dokument auch nach dem Laden manipuliert werden kann. Diese Schnittstelle ist das Document Object Model (DOM).[34] Um Daten von Webservern laden zu können, ohne dabei die gesamte Seite aktualisieren zu müssen, steht die Javascript-Methode *XMLHttpRequest* zur Verfügung, mit der Daten in XML-Form im Hintergrund geladen werden und mithilfe des DOM in die Seite eingesetzt werden können.[35]

3. *XML*, welches wegen seiner Universalität zum Datenaustausch verwendet wird. Neben der Methode *XMLHttpRequest* stellt Javascript noch weitere Methoden zur Verarbeitung von XML bereit.

Da die Umsetzung von RIA mit Ajax die Erstellung vieler komplexer Javascript-Funktionen erfordert, wurden wichtige Methoden von Entwicklergruppen gesammelt, und als so genannte Ajax-Toolkits meist unter Open-Source-Lizenz veröffentlicht. Von diesen Toolkits gibt es eine große Anzahl, die sich in ihrer Qualität und in ihrem Funktionsumfang unterscheiden.[36]

---

33 Vgl. Vossen, Hagemann (2007), S. 46
34 Vgl. Vossen, Hagemann (2007), S. 93
35 Vgl. Vossen, Hagemann (2007), S. 146
36 Vgl. Vossen, Hagemann (2007), S. 211

Daher stellt die Auswahl eines geeigneten Toolkits eine der Herausforderungen bei der Entwicklung dar. Sind die Funktionsumfänge der einzelnen Toolkits noch recht einfach zu vergleichen, stellt die Sicherstellung der Qualität durchaus ein Problem dar, vor allem im Hinblick auf die Kompatibilität mit den benutzten Browsern und Betriebssystemen. Fehler können zum einen bei der Ausführung des Javascripts, zum anderen aber auch bei der Interpretation des HTML-Codes auftreten. Nimmt man dies mit der Anzahl der möglichen Browser/Betriebssystem Kombinationen zusammen, so kann man die Höhe des Testaufwands für Ajax-Anwendungen in etwa abschätzen.

Browser-Plugins

Ein Browser-Plugin ist ein externes Programm, das vom Benutzer selbst in einen Browser integriert werden kann, damit es dort Zusatzfunktionen bereitstellt. Alle gängigen Browser stellen dazu Schnittstellen bereit, welche zur Entwicklung und Einbindung von Plugins verwendet werden können. Eine Kategorie der Browser-Plugins stellen Laufzeitumgebungen dar, die es dem Browser ermöglichen, weitere Inhalte anzeigen zu können, die über das klassische HTML hinausgehen. So existieren z. B. Laufzeitumgebungen zum Abspielen von Audio- und Videodateien, aber auch solche zum Interpretieren und Ausführen von Programmcode. Die Gängigsten und Mächtigsten dieser Art sind die Java Virtual Machine, die Java Applets ausführen kann, sowie der Adobe Flash Player.

Zwischen Java Applets und Flashfilmen der aktuellen Generation[37] können viele Gemeinsamkeiten ausgemacht werden: Beide unterstützen z. B. objektorientierte Programmierung oder bieten Möglichkeiten zur Erstellung grafischer Benutzeroberflächen oder der Benutzung von Netzwerkfunktionen. Dennoch ist die Entwicklung der beiden Ansätze sehr unterschiedlich verlaufen: Java ist ursprünglich eine Plattform zur Entwicklung von Desktopapplikationen. Das Applet-Konzept ist prinzipiell eine Java-Erweiterung, welche die Nutzung dieser Programme im Browser möglich macht. Demgegenüber war Flash ursprünglich nur eine Möglichkeit, Vektorgrafiken und Animationen in Internetseiten integrieren zu können. Die in Flash integrierte Skriptsprache *Actionscript* beinhaltete zu Beginn zumeist Funktionen zur Steuerung der Animationen, wurde aber im Laufe der Entwicklung mit jeder neuen Version des Flash Players mächtiger.

Eine genauere Aufstellung der Funktionsumfänge dieser beiden Ansätze soll im nächsten Abschnitt gegeben werden, in welchem ein Vergleich zwischen Browser-Plugins und Ajax durchgeführt wird.

[37] derzeit Flash Player Version 9 mit ActionScript 3.0

5.2.1.3 Vergleich der Alternativen und Auswahl

Um die Entscheidungsfindung bezüglich der für das Frontend zu nutzenden Plattform zu verdeutlichen, sollen nun die Vor- und Nachteile von Ajax, Flash sowie Applets gegeneinander gestellt und mit den Anforderungen an die Umsetzung des Beergame abgeglichen werden.

Tabelle 5.1 zeigt vorweg das Ergebnis dieser Evaluation, welches dann nachfolgend erläutert werden soll. Die Entwicklung einer klassischen Webanwendung wurde bereits in Kapitel 5.2.1.1 als nicht Ziel führend bewertet und wird damit in dieser Evaluation nicht mehr berücksichtigt.

	Ajax	**Flash**	**Applets**
Performance (Start/Benutzung)	+/-	-/+	-/+
Systemvoraussetzungen	+	-	-
Kompatibilität	-	+	+
Zeichenfunktionen	-	+	+
GUI-Komponenten	+	+	+
Animationen	-	+	-
XML-Funktionen	+	+	+
kostenlose Verfügbarkeit	+	-	+

Tabelle 5.1: Vergleich der Frontend-Alternativen

Der Punkt „Performance“ in der oben stehenden Aufstellung beinhaltet die Ladezeit der Applikation sowie der Belastung von Prozessor und Arbeitsspeicher während des Gebrauchs. Dazu kommt die tatsächliche Ausführungsgeschwindigkeit zentraler Funktionen. Als Ladezeit der Applikation ist zum einen die Zeit zu sehen, die zwischen Aufruf der Seite im Browser und dem vollständigen Laden der Eingangsmaske verstreicht. Zum anderen spielt aber auch die Zeit eine Rolle, die das Programm benötigt, um nach einer Aktion des Benutzers die Anzeige zu aktualisieren. Bei Flash und Applets hängen die anfängliche Ladezeit und weitere Wartezeiten während des Gebrauchs direkt voneinander ab: Der Entwickler kann entscheiden, welche der grafischen Elemente in die Hauptdatei integriert und damit zu Beginn geladen werden, und welche erst bei Bedarf nachgeladen werden. So können Elemente mit großer Ladezeit, etwa aufwendige Animationen, zu Beginn im Hintergrund geladen werden, damit sie bei Bedarf unverzüglich zur Verfügung stehen. Diese

Möglichkeiten stehen bei Ajax nur eingeschränkt zur Verfügung. Die anfängliche Ladezeit ist bei Ajax zwar sehr gering, auch da keine zusätzliche Laufzeitumgebung geladen werden muss, jedoch können grafische Elemente nicht im Voraus geladen werden.

Es ist sicherlich ein Vorteil der Ajax-Technologie, dass außer einem Browser keinerlei zusätzliche Komponenten installiert werden müssen, während Flashfilme das Vorhandensein eines Flash Players in einer bestimmten Version und Applets eine in den Browser integrierte Java-Laufzeitumgebung erfordern. Diese Laufzeitumgebungen sind sowohl für Flash, als auch für Java zwar für alle gängigen Betriebssysteme und Browser verfügbar, werden jedoch teilweise auf betrieblich genutzten Computern oft wegen Sicherheitsbedenken nicht installiert. Dennoch gehört der Flash Player in der Version 7 zum Lieferumfang von Microsoft Windows XP und Studien des Herstellers Adobe geben an, dass auf mehr als 95 Prozent aller zu privaten Zwecken genutzten Rechner ein Flash Player installiert ist.[38] Aufgrund des hauptsächlich lokalen Einsatzes des Beergame sowie der nur geringen Vertraulichkeit der Daten kann zudem von Sicherheitsbedenken abgesehen werden.

Durch die zusätzliche Laufzeitumgebung innerhalb des Browsers kann sichergestellt werden, dass sowohl Java Applets als auch Flashdateien unabhängig von Browser oder Betriebssystem in ähnlicher Qualität funktionieren. Da Ajax-Anwendungen jedoch direkt im Browser ausgeführt werden, hängt deren Qualität direkt von der Kombination Betriebssystem/Browser ab, wobei Unterschiede zum einen in der Interpretation des HTML-Codes als auch bei der Ausführung des Javascripts auftreten können. Durch diese Umstände wird der Testaufwand von Ajax-Anwendungen, aber auch der Aufwand zur Findung eines geeigneten Ajax-Toolkits um ein Vielfaches erhöht.

Im Bereich der Zeichen- und Animationsunterstützung zeigen sich die Stärken der Flash-Plattform, deren Ursprung schließlich die Verarbeitung von Vektorgrafik ist. Es stehen Funktionen zur Erstellung von Formen und Linien sowie zur dynamischen Positionierung von Textelementen bereit. Des Weiteren können sämtliche Elemente des Bildschirms mit Animationen belegt werden, d.h. ein Parameter des Objekts, beispielsweise eine Koordinate, kann in einer bestimmten Geschwindigkeit von einem Wert in einen Anderen überführt werden. Zusätzlich können Spezialeffekte für die Transformation angegeben werden, wie etwa ein stetig langsamer werdendes Einpendeln auf die Zielkoordinate. Wenngleich die Zeichenoperationen auch mit Java Applets in ähnlicher Qualität realisierbar sind, müssten die Animationseffekte manuell durch den Entwickler erstellt werden, was einen erheblich größeren Aufwand zur Folge hätte. Ajax-Entwickler müssen diese Funktionalitäten selbst

[38] Siehe http://www.adobe.com/products/player_census/flashplayer/PC.html

implementieren, oder auf Bibliotheken externer Entwickler zurückgreifen, was die Heterogenität der verwendeten Ajax-Toolkits bzw. Bibliotheken noch weiter erhöht.

Die Verfügbarkeit von Komponenten zur Umsetzung der komplexen Benutzeroberfläche ist bei allen drei Alternativen gegeben. In Java Applets können die bei der Entwicklung von Desktopanwendungen üblichen Bibliotheken AWT und Swing uneingeschränkt verwendet werden. Flash beinhaltet eine Komponentenbibliothek, in der zum einen bereits die gängigsten Steuerelemente im Lieferumfang enthalten sind, die zum anderen aber auch durch eigene Steuerelemente, oder solche externer Entwickler, aufgefüllt werden kann. Bei Ajax hängt der Umfang an Steuerelementen sowie deren Qualität wiederum vom verwendeten Toolkit ab, jedoch sind auch hier die gängigsten Komponenten in den meisten Toolkits enthalten.

Zentraler Bestandteil der Ajax-Technnologie ist die Auswertung von Daten im XML-Format per JavaScript, wobei die Verwendung von XML durch die Universalität und Kompatibilität dieses Datenformats begründet wird. Die Verarbeitung von XML ist jedoch keine Spezialität von JavaScript, sondern wird sowohl von Flash als auch in Java Applets durch die Schnittstellen DOM und XPath unterstützt.[39] Funktionen, die der Ajax-Methode *XMLHttpRequest* gleichkommen, sind ebenfalls vorhanden.

Im Gegensatz zu Java und auch den meisten Ajax-Toolkits baut Flash nicht auf dem Open-Source Gedanken auf, sondern ist ein proprietäres Produkt der Firma Adobe.[40] Demzufolge benötigt man zur Erstellung von Flashdateien eine Entwicklungsumgebung, die von ebendieser Firma kostenpflichtig angeboten wird. Glücklicherweise stellte diese Tatsache für das vorliegende Projekt kein größeres Hindernis dar, da dem Autor für die Dauer der Entwicklung von der reflact AG in Oberhausen freundlicherweise ein Flash-Arbeitsplatz zur Verfügung gestellt wurde.

Aufgrund der Unzulänglichkeiten im Bereich der Animations- und Zeichenfunktionen, die zur Umsetzung der Diagrammen und der animierten Spielansicht unabdingbar sind, scheidet die Ajax-Technologie für eine Umsetzung des Beergame aus. Zudem kann durch das kurze Zeitfenster zur Entwicklung der Anwendung das Risiko eventueller Browserinkompatibilitäten des gewählten Toolkits nicht eingegangen werden. Bei der Abwägung zwischen Java Applets und Flash geben ebenfalls die größeren grafischen Möglichkeiten der Flash-Plattform den Ausschlag, da kein Kriterium ausgemacht werden konnte, dass den Einsatz der einen oder anderen Methode unmöglich machen würde. Sowohl mit Applets als auch mit Flash kann das Beergame adäquat implementiert werden, daher geben die ein-

[39] In Java: org.w3c.dom und org.w3c.dom.xpath
In Flash: XMLNode..XML sowie mx.xpath.XpathAPI

[40] Vormals Macromedia.

fachere Handhabung der Animationsfunktionen sowie die (natürlich vom Entwickler subjektiv eingeschätzt) ansprechende optische Anmutung der Flash-basierten Benutzeroberflächen den Ausschlag.

5.2.2 Datenaustausch und persistente Datenhaltung

Datenaustausch zwischen Client und Server

Im vorherigen Abschnitt wurde dargelegt, dass Flash zahlreiche Methoden zur Verarbeitung von XML-Daten anbietet. Diese sind vor allem ein spezielles Objekt zum Senden und Empfangen von XML Daten über das Netzwerk sowie die bereits erwähnte Implementierung der XPath-Schnittstelle zur Datenextraktion aus XML. Damit bietet Flash die Möglichkeit, Daten, die in Form von XML von einem Server empfangen wurden, ohne weitere Transformationen im Programm zu nutzen.

Ein weiterer Vorteil von XML ist die Möglichkeit, die gewählte Datenstruktur in einer Definitionsdatei (DTD) oder einem XML Schema festzulegen[41]. Ein Server kann empfangene XML-Daten über bereits existierende Validierungsmethoden mit der Definition abgleichen und eine gezielte Fehlerbehandlung durchführen, falls eine Differenz vorliegen sollte.[42] Im Frontend kann zudem das sog. Data Binding genutzt werden: Da die Struktur der empfangenen Daten im Voraus feststeht, können Variablen oder Texte in der GUI direkt mit einem Element eines XML-Fragments verbunden werden.

Die Nutzung von XML in Verbindung mit einer Definitionsdatei als Schnittstelle zum Backend hat weitere entscheidende Vorteile: Zum einen ist XML ein universell lesbares Datenformat und somit nicht an die Nutzung eines Flash-basierten Frontends gebunden. Es besteht ohne weiteres die Möglichkeit, in Zukunft mit einem alternativen Frontend oder externen Programmen auf die Programmlogik des Beergame zuzugreifen.

---

41 Vgl. Lehner, Schöning (2004), S.22 ff.

42 z.b in Java mithilfe der Bibliothek javax.xml.validation

Persistente Datenhaltung

Zur dauerhaften Speicherung der anfallenden Daten stehen mehrere Möglichkeiten zur Auswahl:

1. die Sicherung der Daten in einer relationalen Datenbank,
2. die Speicherung proprietärer Dateien im Dateisystem,
3. oder die Speicherung im Dateisystem, jedoch in einem universellen Datenformat.

Die Nutzung von Datenbanken ist von daher immer zu bedenken, da sie durch die normalisierte Darstellung der Daten eine sehr effiziente Methode der Datensicherung darstellt, die zudem ein hohes Maß an Transaktionssicherheit aufweist. Bei der vorliegenden Problemstellung sprechen allerdings zwei Punkte gegen die Nutzung einer relationalen Datenbank: Erstens ist eines der Hauptziele der Entwicklung die Sicherstellung einer einfachen Installation. Da relationale Datenbanken jedoch die zusätzliche Installation eines Datenbankmanagementsystems (DBMS) erfordern, sollte von dieser Methode Abstand genommen werden. Des Weiteren ist absehbar, dass die anfallende Datenmenge und die Anzahl der Transaktionen während des Betriebs wohl zu gering sind, um den Einsatz eines DBMS zu rechtfertigen.

In objektorientierten Programmiersprachen, wie z. B. Java, ist es möglich, Objektinstanzen auf dem Dateisystem abzulegen und zu einem späteren Zeitpunkt als vollwertige Instanz wieder herzustellen.[43] Da dazu keinerlei Transformation der Daten notwendig ist, findet somit durch das Speichern und Laden kein Informationsverlust statt. Allerdings sind auch bei dieser Methode zwei Nachteile zu nennen: Erstens sind die erstellten Dateien an die Programmiersprache und das Klassenmodell gebunden. Mit Java erstellte Dateien können somit auch nur von Java-Programmen gelesen werden und auch nur von solchen, welche die Ursprungsklassen der gesicherten Dateien kennen. Somit würde eine zusätzliche Barriere geschaffen, was den Zugriff externer Programme auf Daten des Beergame betrifft.

Es empfiehlt sich daher, auch bei der dauerhaften Datensicherung auf ein universelles Datenformat zurückzugreifen. Da XML ein solches ist und bereits für den Datenaustausch als vorteilhaft erachtet wurde, bietet sich an, dieses Format auch zur Datensicherung zu verwenden.

---

[43] In Java durch die Implementierung des Interface java.io.Serializable sowie der Verwendung eines java.io.ObjectOutputStream.
Vgl Riley (2002), S.508

Als Ziele für die nächsten Abschnitte können demnach identifiziert werden:

1. Eine Serverplattform zu wählen, welche die Speicherung von Dateien im Dateisystem des Servers erlaubt und Methoden zur Erstellung, Manipulation und Validierung von XML-Daten bereitstellt.

2. Methoden zu implementieren, die aus bestehenden Zuständen des Beergame-Backend XML-Repräsentationen generieren, sowie im Umkehrschluss neue Instanzen aus einer zuvor gesicherten XML-Datei erzeugen können.

5.2.3 Technologien für das Backend

Nimmt man die Anforderungsanalyse als Grundlage und berücksichtigt die bisherigen Entscheidungen bezüglich der Softwarearchitektur, so lassen sich diejenigen Funktionen identifizieren, die vom Backend des Beergame unterstützt werden müssen. Diese sollen nachstehend erläutert werden.

Die Hauptfunktion des Backends kann unter dem Oberbegriff „Datenmanagement“ zusammengefasst werden. Dazu gehört zum einen die permanente Speicherung der Daten in verschiedener Form und zum anderen die Synchronisierung der Datenbestände der einzelnen Clients. Voraussetzung für beide Funktionen ist zunächst die Zusammenführung aller während des Spiels anfallenden Daten.

Direkt zusammenhängend mit dem Bereich der Datensynchronisation der Clients ist die Frage, an welcher Stelle in der Softwarearchitektur die Spiellogik zu finden ist. Hier gibt es zwei Möglichkeiten:

1. Der Server sorgt dafür, dass alle Clients auf demselben Informationsstand sind, indem er Eingaben, also Bestellungen, eines Spielers an alle anderen Spieler weiterleitet. Die Berechnung aktueller Rundendaten, wie beispielsweise der Wareneingang oder der eingehende Kundenauftrag werden clientseitig in Flash berechnet.

2. Der Server sammelt die Eingaben der Spieler und berechnet den neuen Datenbestand des Spiels. Den Clients werden nur genau die Informationen zugeführt, die auf der Benutzeroberfläche angezeigt werden sollen. Berechnungen finden damit nur zentral auf dem Server statt.

Variante 2 ist aus mehreren Gründen als vorteilhaft anzusehen: Ein Grund ist die bereits angesprochene Vorgabe, den aktuellen Stand sowie den gesamten Verlauf eines Spiels

speichern zu können. Dies wird erheblich vereinfacht, wenn alle dazu benötigten Informationen an einer Stelle zusammenlaufen. Ein weiterer Grund ist die sich dadurch ergebene Unabhängigkeit von einem bestimmten Frontend. Befinden sich sowohl Spieldaten als auch Spiellogik zentral auf dem Server, so kann ohne Weiteres ein alternatives Frontend erstellt werden, welches mit dem Backend des Beergame kommunizieren kann, solange es die zuvor festgelegte XML-Schnittstelle beachtet. Auch ein völlig anders gearteter Zugriff, beispielsweise über statistische Analysetools ist möglich. In Kapitel 7 wird näher auf diese Möglichkeiten eingegangen.

In Kapitel 5.2.2 wurde erarbeitet, dass das Backend Methoden zur Verarbeitung und Validierung von XML bereitstellen soll. Neben der Datensicherung in XML soll zudem die Möglichkeit bestehen, die Daten für Tabellenkalkulationsprogramme lesbar zu machen. Die eleganteste Lösung ist sicherlich, die Spieldaten direkt in ein Excel-Spreadsheet zu exportieren. Dazu muss eine Möglichkeit gefunden werden, Spreadsheet-Dateien zu erstellen, bzw. Daten in eine bestehende Vorlage einfügen zu können.

Zusammenfassend wurden für das Backend die folgenden Aufgaben identifiziert:

- Speichern/Laden von Dateien im Dateisystem
- Unterstützung von XML
- Zugriff auf Excel-Spreadsheets
- Bereitstellung der Spiellogik und objektorientierte Abbildung des Datenmodells
- Die Annahme und Beantwortung von Anfragen der Clients

Zur Erfüllung dieser Anforderungen bietet sich die Benutzung der Java-Plattform an. Java ist eine objektorientierte Programmiersprache, die jedem Entwickler kostenfrei zur Verfügung steht. Bibliotheken mit Methoden zur Verarbeitung von XML und dem Zugriff auf das Dateisystem sind bereits in Java enthalten. Zudem werden von der Apache Foundation Bibliotheken zum Zugriff auf Excel-Spreadsheets angeboten, die ebenfalls kostenlos und einschränkungsfrei nutzbar sind.[44]

Zur Beantwortung von Clientanfragen stellt Java zwei Methoden zur Verfügung: Java Server Pages und Servlets. Beide Konzepte wurden bereits zur Umsetzung des Frontends erläutert und würden hier in ähnlicher Weise genutzt. Zuvor ging es darum, serverseitig HTML-Code zusammenzustellen und an den Browser zu senden. Nun kommen die Anfra-

[44] Siehe Apache POI Project, poi.apache.org.

gen aus den Flash-Filmen und erwarten XML als Rückgabe. Die Verwendung von Servlets erleichtert zudem die Anwendung des Model-View-Controller (MVC) Konzepts. Bei MVC geht es darum, zwischen dem Frontend und der Programmlogik eine weitere Ebene einzufügen. An diese werden alle Aufrufe des Frontends gerichtet und nur aus dieser Ebene werden Aufrufe der Programmlogik vorgenommen.[45] Diese Ebene wird in diesem Muster Controller genannt und ist zwischen Frontend (View) und Programmlogik (Model) angeordnet. Der Controller übernimmt die Annahme sowie die Validierung der Anfragen. Nur gültige Anfragen werden an die Programmlogik weitergeleitet, so bleiben die laufenden Spiele stets in einem konsistenten Zustand. Mittels Servlets kann MVC sehr einfach umgesetzt werden. Wird für sämtliche Server-Aufrufe nur ein Servlet genutzt, so laufen alle Anfragen in einer Methode des Servlets zusammen. Die Aufrufe können daraufhin durch ihre Parameter unterschieden und überprüft werden. Der genaue Ablauf einer solchen Anfrage wird in Kapitel 6.1.2 näher erläutert.

Um Java, und im Speziellen Servlets, als Grundlage für einen Webserver zu benutzen, ist eine Server-Software notwendig, die eine Laufzeitumgebung für Servlets, sog. Servlet-Container, bereitstellt. Weitere Anforderungen an die Server-Software sind eine möglichst einfache Installation sowie eine möglichst kostenfreie Bereitstellung.

Diese Anforderungen erfüllt der Apache Tomcat. Tomcat ist ein Open Source Projekt der Apache Foundation und ist zum Zeitpunkt der Entwicklung in der Version 5.5 stabil für die Plattformen Windows, Linux und Mac OS X verfügbar.[46] Die Nutzung des Tomcat ist kostenlos. Zudem ist Tomcat vollständig in Java entwickelt worden, benötigt somit nur eine Java Laufzeitumgebung und muss nicht gesondert installiert werden. Dieser Umstand bietet dem Spielleiter eine höchstmögliche Flexibilität beim späteren Gebrauch, beispielsweise können der Tomcat und die Java-Laufzeitumgebung von einem USB-Speicherstick geladen werden. Der Spielleiter kann dann jeden beliebigen Computer im Netzwerk zum Beergaqme-Server machen, in dem der Tomcat direkt vom Stick auf diesem Rechner gestartet wird. Dazu sind keinerlei weitere Anpassungen des Computers nötig.

Abb. 5.2 zeigt abschließend eine weitere grafische Darstellung der in diesem Kapitel erarbeiteten Softwarearchitektur. Sie enthält ein detailliertes Bild der vorliegenden Client/Server Architektur, welches nun die gewählten Laufzeitumgebungen enthält sowie die Konzepte zum Datenaustausch zwischen Client und Server und zur Datensicherung.

---

45 Vgl. Gamma, Helm, Johnson, Vlissides (2004), S.5 ff.

46 Siehe Apache Tomcat Project, tomcat.apache.org

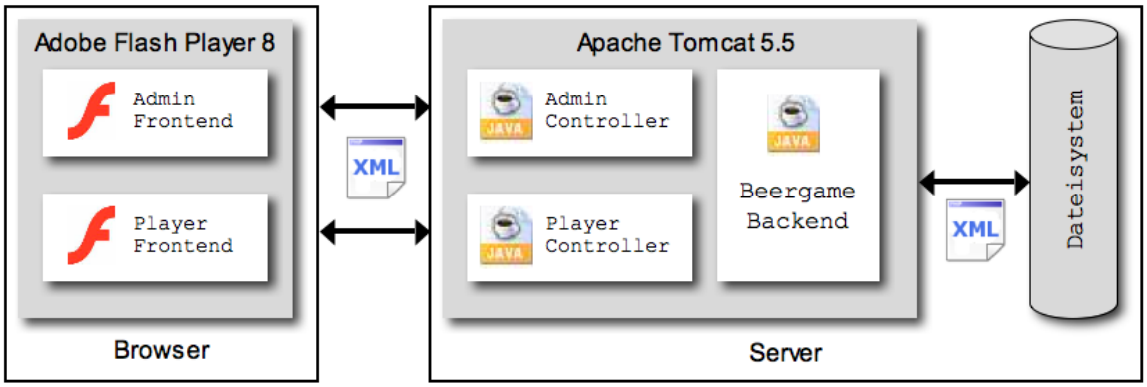


Abb. 5.2: Erweiterte Softwarearchitektur

Nachdem nun alle zentralen Entscheidungen getroffen wurden, die vor der eigentlichen Umsetzung der softwaretechnischen Realisierung des Beergame getroffen werden mussten, kann nun auf Grundlage der erarbeiteten Softwarearchitektur die Implementierung der Software beginnen. Einen Überblick über die Herausforderungen der Implementierungsphase wird das nächste Kapitel geben.

6 Implementierung

6.1 Backend

Ziel dieses Kapitels soll es nicht sein, die Umsetzung jeder einzelnen Klasse durch Quellcode zu beschreiben. Vielmehr sollen diejenigen Aspekte erläutert werden, die während der Planungsphase noch nicht abgesehen werden konnten, wie etwa solche, die eng an Konzepte der Programmiersprache gebunden sind.

Abb. 6.1 zeigt die vollständige Aufzählung aller im Backend verwendeten Java-Klassen nach Beendigung der Implementierungsphase sowie deren Aufteilung in Java-Pakete. In den nächsten Abschnitten werden nun wiederholt einzelne Klassen oder Pakete herausgenommen und deren Umsetzung dargelegt.

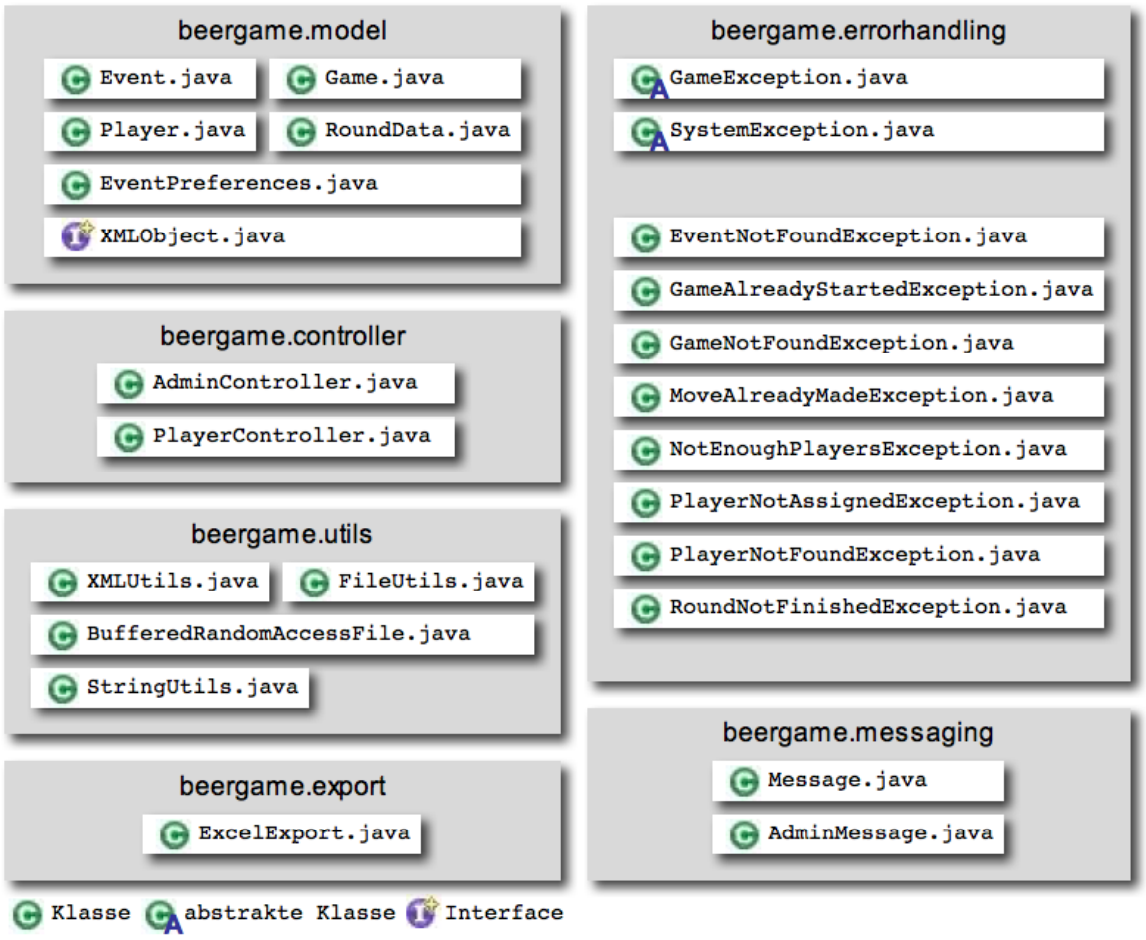


Abb. 6.1: Paketstruktur der Java-Klassen des Backend

6.1.1 XML-Objekte

Wie bereits erläutert, wird zum Datenaustausch zwischen Frontend und Backend sowie zur dauerhaften Datenhaltung das XML-Format verwendet. Dies hat zur Folge, dass eine Möglichkeit geschaffen werden musste, Instanzen verschiedener Klassen des Backends in XML umwandeln zu können. Dies betrifft alle Klassen der Pakete *beergame.model* und *bergame.messaging*. Um bestehende Konfigurationen oder Veranstaltungen von der Festplatte laden zu können ist es zudem notwendig, aus XML-Dateien neue Objektinstanzen initialisieren zu können.

Um diese Funktionen bereitzustellen, wurde das Interface *XMLObject* eingeführt, welches sich im Paket *beergame.model* befindet und von den oben genannten Klassen implementiert wird. Kern des Interface *XMLObject* ist die Methode *toXML(int depth)*. Diese Methode liefert eine XML-Repräsentation des Objekts als Zeichenkette zurück. Die explizite Zusammenstellung des XML-Strings wird in jeder Klasse gesondert implementiert. Der Parameter *depth* gibt die Tiefe der XML-Verschachtelung an. Dies funktioniert folgendermaßen: Befinden sich in dem betreffenden Objekt Referenzen auf andere Objekte, die auch das Interface *XMLObject* implementieren, so wird während des Zusammenstellens der XML innerhalb der Methode *toXML(int depth)* dieselbe Methode auch in den untergeordneten Objekten aufgerufen, jedoch mit einer um Eins dekrementierten Tiefe. In den toXML-Methoden der Unterobjekte wird analog verfahren, solange die übergebene Tiefe größer ist als Null.

Die Methode *toXML* wird zum einen von den Controllern aufgerufen, um das Frontend in geeigneter Form mit der aktuellen Datenlage des Beergame zu versorgen. Zum anderen wird bei jedem Rundenübergang der aktuelle Stand eines Spiels gesichert, indem die Methode *toXML* in der Klasse *Game* mit maximaler Tiefe aufgerufen wird. Das Ergebnis wird in der Datei *gameDump.xml* im Ordner des Spiels auf dem Server gesichert.

Zusätzlich zu der oben beschriebenen Methode *toXML* enthalten alle *XMLObjekt* implementierenden Klassen einen zusätzlichen Konstruktor, welcher die Attribute eines bestehenden Objekts aus einem XML-Dokument füllen kann. Im Falle der Klasse *EventPreferences* wird dieser Konstruktor von zwei statischen Factory-Methoden[47] verwendet, mit deren Hilfe eine neue Instanz der Klasse entweder aus einer Datei auf der Festplatte, oder aus einer XML-formatierten Zeichenkette erzeugt werden kann. Diese Methoden werden

[47] Vgl. Gamma, Helm, Johnson, Vlissides (2004), S.62 ff.

dazu genutzt, gespeicherte Konfigurationen zu laden oder zu speichern, oder die aktuelle Konfiguration durch eine vom Frontend kommende XML-Zeichenkette anzupassen.

6.1.2 Anfragen an die Controller

Anfragen des Frontends an das Backend laufen stets nach demselben Muster ab: Der Client ruft eine URL auf, die auf eines der beiden Controller-Servlets zeigt. In dieser URL müssen alle für den Aufruf benötigten Parameter enthalten sein.[48] Sind die Parameter vollständig und vom Backend als gültige Werte bestätigt worden, wird das Ergebnis generiert und in Form von XML an das Frontend zurückgesendet.

Zentraler Parameter eines jeden Controller-Aufrufs ist der Parameter *action*. Durch den Wert von *action* kann der Controller die Art der Anfrage identifizieren. Ein häufig benutzter Wert für *action* ist beispielsweise „player_join_event“, wodurch eine Anfrage zum Hinzufügen eines Spielers zu einer Veranstaltung gekennzeichnet wird.

In Abhängigkeit von *action* werden anschließend sämtliche für die Anfrage benötigten Parameter abgefragt. Im genannten Beispiel „player_join_event“ sind dies der Name des Spielers sowie dessen Email-Adresse. Sind einer oder mehrere dieser Parameter nicht spezifiziert worden oder enthalten unzulässige Werte, so wird eine entsprechende Exception ausgegeben und die Bearbeitung der Anfrage abgebrochen. Die genaue Funktionsweise des Exception-Handling im Beergame wird im nächsten Abschnitt erläutert. Eine vollständige Aufstellung aller möglichen Controller-Aufrufe befindet sich im Anhang dieser Arbeit. Die Aufstellung enthält für jede *action* die weiteren benötigten Parameter mit Informationen über die gültigen Wertebereiche der Parameter. Zudem werden die XML-Struktur der Antwort sowie die möglichen Exceptions angegeben. Die nachstehende Grafik zeigt exemplarisch eine URL für die Abgabe einer Bestellung durch einen Spieler.

Abb. 6.2: URL zur Abgabe einer Bestellung

Die Verwendung von URL-Parametern muss bei der Implementierung des Backend speziell berücksichtigt werden. Das liegt vor allem daran, dass eine URL, und damit auch die

[48] Vgl. Loius, Müller (2005), S.924

darin enthaltenen Parameter, nur als Zeichenkette zur Verfügung steht. Daher muss für jeden Datentyp, der als Parameter übergeben wird, eine String-Repräsentation gefunden werden, aus der das Backend den ursprünglichen Datentyp wiederherstellen kann. Für etliche Datentypen sind diese Repräsentationen bereits vorhanden. Als Beispiel sei hier die Übergabe von Zeit- bzw. Datumsangaben genannt. In vielen Programmiersprachen wird die Benutzung von sog. Zeitstempeln unterstützt, beispielsweise vom Datentyp *Date* in Flash oder von *java.util.Date* in Java. Unter einem Zeitstempel versteht man eine Zahl, welche die Differenz zwischen dem angegebenen Datum und dem 1. Januar 1970 in Millisekunden darstellt.[49] Zu bestehenden Datumsobjekten können Zeitstempel berechnet werden, zudem können aus Zeitstempeln neue Datumsobjekte erzeugt werden. Die Umwandlung von Zahlen in Strings zur Verwendung als URL-Parameter und umgekehrt ist danach problemlos.

Nicht so trivial wie die Übergabe von Datumsangaben ist der Verweis auf bestehende Objekte im Backend. Sollen beispielsweise Informationen über eine bestimmte Veranstaltung abgefragt werden, so verlangt das Frontend nach einer ganz bestimmten Instanz der Klasse *Event*. Es musste eine Möglichkeit geschaffen werden, diese Instanz aufzufinden, unter der Einschränkung, dass das Frontend keine direkte Objektreferenz vorhält, sondern nur Strings zur Identifizierung der Instanz übergeben kann. Dazu wurden für die Klassen *Event*, *Game* und *Player* Schlüsselattribute angelegt, wie es ähnlich auch bei relationalen Datenbanken bekannt ist. Zudem gibt es in jeder dieser Klassen Methoden, die zu einem gegebenen Schlüssel die entsprechende Instanz zurückliefert.

Die verwendeten Schlüssel müssen zwei Anforderungen genügen: sie müssen ein Objekt eindeutig identifizieren und sie müssen in Form eines Strings vorliegen oder zumindest einfach in einen solchen umgewandelt werden können. Diesen Anforderungen entspricht eine für jedes Objekt eindeutige Nummer, die problemlos in einen String konvertiert werden kann.

Die Klassen *Event*, *Game* und *Player* enthalten die Attribute *eventKey*, *gameKey* und *playerKey*, die im Zahlenformat *long* vorliegen und beim erstellen einer Instanz mit dem Zeitstempel der aktuellen Systemzeit befüllt werden. Auf diese Weise wird eine Eindeutigkeit der Schlüssel sichergestellt. Die Klassen enthalten weiterhin die Methode *getByKey(long key)*, welche die entsprechende Instanz zurückgibt bzw. eine Exception wirft, falls keine Instanz mit diesem Schlüssel gefunden werden konnte.

Es ist zu beachten, dass sich die Methoden zum Auffinden beispielsweise eines Spielers nicht in der Klasse *Player* befinden, sondern in der nächst höheren Hierarchiestufe des

[49] Vgl. Louis, Müller (2005), S.689 f.

Datenmodells, der Klasse *Game*. Damit kann sichergestellt werden, dass ausschließlich in der Menge an Spielern gesucht wird, die diesem Spiel zugeordnet wurden. Selbes gilt analog für die Klassen *Game* und *Event*. Die Methoden zum Auffinden einer *Game*-Instanz befinden sich als statische Methoden in der Klasse *Event*.

Die Klasse *Player* verfügt neben der Identifikationsnummer über einen weiteren Schlüssel. Da die Eindeutigkeit der Kombination aus Benutzername und Email-Adresse verlangt wird, kann die *Player*-Instanz eines Spielers über diese beiden Werte eindeutig identifiziert werden. Daher verfügt die Klasse Game zusätzlich über die Methode *getPlayer(String username, String email)*. Prinzipiell hätte dieser Schlüssel zur Identifikation einer *Player*-Instanz ausgereicht, aus Gründen der Einfachheit wurde aber auch in der Klasse *Player* die Identifikationsnummer vorgesehen.

6.1.3 Exception-Handling

Die Benutzung von Exceptions zur Fehlerbehandlung ist eines der zentralen Konzepte von Java.[50] Exceptions können dazu genutzt werden, Fehler, die während der Laufzeit des Programms auftreten, zu erkennen und entsprechend darauf zu reagieren. Zu diesen Fehlern gehören beispielsweise ungültige Benutzereingaben, es können aber auch inkonsistente Programmzustände abgefangen werden, die aus Fehlern im Programm selbst resultieren. Somit stellen Exceptions ein wichtiges Feedbackmedium sowohl für Entwickler als auch für Benutzer dar.

Bei der Frage, in welcher Form die Rückmeldung nach Auftritt einer Exception geschehen soll, muss nach Art des Fehlers unterschieden werden. Systemfehler, also Fehler, die durch mangelhafte Implementierung hervorgerufen wurden, sind vor allem für den Entwickler relevant. Da der Entwickler naturgemäß nicht bei jeder Veranstaltung vor Ort sein kann, musste eine Möglichkeit geschaffen werden, auftretende Fehler hinreichend aussagekräftig zu dokumentieren und dauerhaft zu sichern. Zu diesem Zweck wurde bei der Implementierung auf das Java-Logging-Framework *log4J* zurückgegriffen.[51] Log4J bietet die folgenden Möglichkeiten:

1. Textausgaben, die einer von drei Fatalitätsgraden zugeordnet wurden („Debug“, „Info“ oder „Error“), können in den Java-Code integriert, und im Bedarfsfall ausgegeben werden.

[50] Vgl. Louis, Müller (2005), S. 415 ff.
[51] Siehe logging.apache.org/log4j/docs/

2. Ausgaberegeln für jedes dieser Grade zu definieren. Beispielsweise die Sicherung von „Error“-Meldungen in Protokolldateien, jedoch die Ausgabe von „Debug“-Meldungen ausschließlich in der Java-Konsole.

3. Die Fehlermeldungen können um zusätzliche Daten erweitert werden, wie etwa der exakten Zeit des Fehlerauftritts. So kann das spätere Auffinden der Fehlerursache erleichtert werden.

Neben der Benutzung von log4J zum Festhalten von Fehlern musste außerdem eine Möglichkeit geschaffen werden, den Benutzer über aufgetretene Programmfehler zu informieren. Auch bei der Behandlung falscher Eingaben ist eine Rückmeldung an den Benutzer unerlässlich. Bei der Rückgabe einer Exception an das Frontend mussten zwei Aspekte ermöglicht werden: Zum einen die Übersetzung der Exception in ein einheitliches XML-Format, zum anderen muss die Art des Fehlers vom Frontend eindeutig identifizierbar sein, um ggf. eine spezielle Fehlerbehandlung ausführen zu können.

Zur Sicherstellung der beiden genannten Punkte wurde die abstrakte Klasse *GameException* eingeführt, welche die Oberklasse für alle im Beergame vorkommenden Exceptions darstellt. Diese Klasse enthält die Attribute *errorKey* und *details*, die von der Unterklasse gesetzt werden. Dabei enthält *errorKey* den geforderten eindeutigen Identifizierungsschlüssel, in *details* kann eine Fehlerbeschreibung abgelegt werden. Tritt ein Fehler auf, so kann wiederum über die Methode *toXML()* die XML-Repräsentation des Fehlers ausgegeben werden, welche für alle Fehler die gleiche Struktur aufweist sowie die Parameter *errorKey* und *details* enthält.

6.1.4 Export der Spieldaten als Excel-Spreadsheet

Beim Beergame wird nach Spielende für jedes Spiel eine Datei im Microsoft Excel Format angelegt, die sich neben den XML-Daten im Ordner des Spiels befindet. Diese Datei besteht aus mehreren Tabellenblättern: ein Titelblatt mit allgemeinen Informationen über Veranstaltung und Spiel sowie je ein Tabellenblatt pro Spieler. Der Inhalt eines Spieler-Tabellenblatts ist praktisch identisch mit dem Inhalt der Rundendaten-Tabelle im Frontend des Spielers[52], wobei die Hintergrundfarben der Tabellenblätter zur besseren Unterscheidung der einzelnen Supply-Chain Stufen variieren.

Zur Erstellung von Excel-Dateien wurden die frei verfügbaren Java-Bibliotheken des Apache POI Frameworks verwendet.[53] Diese Bibliotheken enthalten Methoden, mit denen es

[52] Siehe hierzu Absatz „Tabellenansicht“ in Kapitel 8.2
[53] Siehe http://poi.apache.org

recht komfortabel möglich ist, Excel-Dateien entweder von Grund auf neu zu erstellen oder aber bereits existierende Dateien zu öffnen und diese dann zu verändern. Im vorliegenden Fall wurde letztere Variante gewählt, da es so möglich war, sämtliche Formatierungen des Dokuments, vor allem Schriftarten, Hintergrundfarben und Zellumrandungen bereits in einem Vorlage-Dokument vorzukonfigurieren.[54] Das äußerst umständliche Formatieren von Tabellen per Java-Code kann somit entfallen. Bei jedem Export wird unter Verwendung die POI-Bibliotheken die Vorlage kopiert und mit den entsprechenden Daten gefüllt.

6.2 Frontend

6.2.1 Verbindung zum Server und Aktualisierung in Echtzeit

Um XML-Daten von einem Webserver senden und empfangen zu können, steht in Flash ein spezielles XML-Objekt zur Verfügung.[55] Darin wird das geladene XML direkt in eine Objektstruktur überführt, durch die anschließend auf die Daten zugegriffen werden kann. Falls das XML nicht vollständig vom Server geladen, oder durch falsche Formatierungen nicht in das genannte Objekt konvertiert werden konnte, werden vom XML-Objekt entsprechende Ereignisse ausgelöst, die der Entwickler berücksichtigen kann.

Ähnlich zum *XMLHttpRequest* bei Ajax erfolgt das Laden der Daten asynchron, d. h. die *load()*-Methode des Objekts wird sofort nach dem Abschicken der Anfrage an den Server beendet. Sind die Daten vollständig geladen, wird diejenige Methode aufgerufen, die mit dem *onLoad*-Ereignis des XML-Objekts verknüpft wurde. Das sofortige Beenden der *load()*-Methode birgt sowohl Vor- als auch Nachteile. Als vorteilhaft ist anzusehen, dass das Programm während des Ladens normal weiterläuft, statt durch eine langsame Netzwerkverbindung oder die Übertragung größerer Datenmengen blockiert zu werden. Ein für die Programmgestaltung nicht unerheblicher Nachteil ist aber, dass ursprünglich zusammengehörige Programmteile in unterschiedliche Methoden verteilt werden müssen. Alle Anweisungen, die erst nach vollständigem Empfang der Daten ausgeführt werden dürfen, können nicht direkt unterhalb der *load()*-Anweisung platziert werden, sondern müssen in der *onLoad()*-Methode ausgeführt werden. Dadurch nimmt die Komplexität des Quellcodes zu, die Übersichtlichkeit des Codes leidet.

---

54 Die Vorlagedatei „export_template.xls“ befindet sich im Hauptordner des Beergame-Backend.

55 Vgl. Macromedia (2005)a, S.1285 ff.

Um die Befehle zur Anfrage an den Server an genau einer Stelle im Code zu kapseln und dennoch eine höchstmögliche Flexibilität bei der Verwendung von Serveraufrufen zu gewährleisten, wurde im Frontend des Beergame wie folgt vorgegangen:

Die einzige Methode, die Anfragen an den Server stellen kann, ist die Methode *xmlLoad*, deren Hauptbestandteil ein XML-Objekt ist. Diese Methode verlangt als Parameter zum einen die vollständige URL des Aufrufs, sowie den Verweis auf eine Funktion, die nach erfolgreichem Laden der XML aufgerufen werden soll. Die Möglichkeit, Funktionszeiger zu setzen ist eine Eigenheit des Actionscript, die in diesem Falle sehr nützlich ist. In der *onLoad*-Methode des XML-Objekts wird nach Ankommen der XML zunächst geprüft, ob es sich um die XML-Repräsentation einer Exception handelt. Ist dies der Fall, wird die XML an die Fehlerbehandlung weitergeben. Falls nicht, wird die zuvor übergebene Methode aufgerufen und das XML-Objekt als Parameter übergeben.

Da die Methode *xmlLoad* eine URL verlangt und es aus Gründen der Fehlervermeidung nicht sinnvoll ist, an vielen verteilten Stellen des Codes die Zusammenstellung der URL vorzunehmen, wird die Methode *xmlLoad* nie direkt vom Programm aus aufgerufen. Stattdessen gibt es für jeden möglichen Controller-Aufruf eine eigene Methode, welche die vom Controller benötigten Parameter verlangt, daraus die URL zusammenstellt und schließlich die *xmlLoad*-Methode aufruft. Alle diese Methoden befinden sich an derselben Stelle des Quellcodes, sodass Änderungen, die alle Aufrufe betreffen, einfach vorgenommen werden können.

Zum genauen Ablauf einer Anfrage nun ein kurzes Beispiel: Nach dem Start des Spieler-Frontends erscheint die Anmeldemaske, in welche der Spieler seinen Namen und seine Email-Adresse eingibt sowie aus der Dropdown-Liste eine Veranstaltung auswählt. Um diese Liste mit den aktuell auf dem Server laufenden Veranstaltungen zu füllen, muss vom Server eine Liste aller Veranstaltungen abgefragt werden. Der dazu passende *action*-Parameter für den *PlayerController* lautet „event_getall“, die Zusammenstellung der URL und der Serveraufruf wird von der Methode *getEventList* übernommen. Beim Anzeigen der Anmeldemaske wird nun die Methode *getEventList* aufgerufen und als Parameter ein Verweis auf die Methode *fillCombo* übergeben. Nach erfolgreichem Empfang der Liste wird automatisch *fillCombo* mit der XML der Veranstaltungsliste aufgerufen und innerhalb dieser Methode die Dropdown-Liste gefüllt.

Anhand dieses Beispiels kann noch ein weitere Funktionalität des Frontends erläutert werden: die automatische Aktualisierung der Daten auf dem Bildschirm ohne weiteres Zutun des Benutzers. Dazu muss, um im Beispiel zu bleiben, die Abfrage der Veranstaltungsliste sowie das Füllen des Dropdowns in regelmäßigen Abständen wiederholt werden. Ist das Intervall hinreichend kurz, kann durchaus von einer Aktualisierung in angenäherter Echt-

zeit gesprochen werden. Hierbei muss natürlich ein Kompromiss zwischen der Aktualität der Daten und der Belastung des Servers gefunden werden. Für gewöhnlich betragen die Aktualisierungsintervalle im Frontend zwei Sekunden. Für das kontinuierliche Ausführen von Methoden bietet Flash die Methode *setInterval*, der zum einen die aufzurufende Methode, sowie die Intervalllänge in Millisekunden übergeben wird.[56] Ein gesetzter Intervallaufruf kann mithilfe von *clearInterval* wieder beendet werden.

Aus Gründen der Performanz wurde darauf geachtet, zwei Regeln konsequent umzusetzen:

1. Nach dem Empfang neuer Daten werden diese zunächst mit den bisherigen Daten verglichen. Nur falls Änderungen vorliegen, wird der Bildschirm aktualisiert. Aus diesem Grund muss stets eine Kopie der empfangenen Daten zum Vergleich gesichert werden.

2. Aktualisierungsintervalle laufen nur dann, wenn eine Anpassung des Bildschirms auch möglich ist. Beispielsweise muss die Spielansicht (Animation oder Tabelle) im Spieler-Frontend nur dann aktualisiert werden, wenn der Spieler bereits seine Bestellung abgegeben hat und auf die nächste Runde wartet. Hat die nächste Runde begonnen und dem Spieler liegen die aktuellen Daten vor, so kann die Aktualisierung bis zur Abgabe der Bestellung ausgesetzt werden.

6.2.2 Verwendung der XML-Daten

Im vorherigen Abschnitt wurde dargelegt, dass die XML-Daten nach erfolgreichem Empfang an eine dafür spezifizierte Methode übergeben werden, in der sie anschließend ausgewertet werden können. Hauptsächlich sind dies Methoden, die anhand der empfangenen Daten Bereiche der Benutzeroberfläche aktualisieren. Methoden dieser Art heißen im Frontend des Beergame oft *fillTable, fillCombo* oder *updateGUI*.

Die übergebenen XML-Daten liegen in Form eines Objekts vom Typ XML vor. Dieser Datentyp wurde zuvor bereits zum Empfang der Daten benutzt. Um nun auf die Inhalte dieses XML-Objekts zugreifen zu können, bietet Flash im Wesentlichen zwei Möglichkeiten an:

1. Die Navigation durch die Elementhierarchie des XML-Dokuments mittels des Document Object Models (DOM).[57]

2. Die direkte Extraktion der gewünschten Elemente über die XPath-Schnittstelle.[58]

---

56 Vgl. Macromedia (2005)a, S.96
57 Vgl. Klettke, Meyer (2003), S.46 ff.

Mit beiden Verfahren ist der Zugriff auf sämtliche Informationen möglich, die in einem XML-Dokument enthalten sind. Der wesentliche Unterschied zwischen beiden Ansätzen liegt darin, dass es sich bei DOM um eine standardisierte Datenstruktur für XML-Dokumente handelt, bei der über festgelegte Methoden auf Elemente und Attribute des Dokuments zugegriffen werden kann. Demgegenüber ist XPath abfragebasiert, d.h. die Datenextraktion erfolgt über Abfrage-Strings, die einer gewissen Syntax genügen müssen.

Dass bei der vorliegenden Umsetzung auf XPath zurückgegriffen wurde, ist hautsächlich dessen komfortablerer Handhabung zuzuschreiben. Für die meisten Abfragen ist mit XPath nur genau ein Methodenaufruf notwendig, wohingegen DOM die „manuelle“ Navigation durch das Dokument verlangt. Zudem ist einem XPath-Aufruf durch den übergebenen Abfrage-String meist auf einem Blick anzusehen, welche Daten die Abfrage liefern wird. Bei DOM ist die Lesbarkeit des Quellcodes weniger intuitiv, was natürlich eine subjektive Präferenz des Entwicklers darstellt.

Im Folgenden soll an einigen prägnanten Beispielen die Benutzung von XPath verdeutlicht werden. Dazu diene der folgende XML-Ausschnitt als Grundlage:

```
<event>
     <title>Vorlesung IOS 23.07.07</title>
     <game>
          <player level="Grosshändler">
              <name>p1</name>
          </player>
          <player level="Spediteur">
              <name>p1</name>
          </player>
     </game>
</event>
```

Die Xpath-API von Flash bietet die Methoden *selectSingleNode* zur Abfrage eines einzelnen XML-Elements, sowie *selectNodeList*, falls das Ergebnis mehrere Elemente umfasst. Diese werden in den folgenden Abfragen genutzt.[59]

1. *Abfrage eines Elements*. Die folgende Abfrage liefert den Titel der Veranstaltung:

   ```
   XPathAPI.selectSingleNode(xmlObj, "/event/title").nodeValue()
   ```

2. *Abfrage einer Liste von Elementen.* Um alle vorhandenen Spieler auszulesen, kann durch den folgenden Aufruf die Liste aller Spieler der Veranstaltung in ein Array geladen werden:

---

58 Vgl. Klettke, Meyer (2003), S.156 ff.
59 Vgl. Macromedia (2005)b, S.3 f.

```
XPathAPI.selectNodeList(xmlObj, "/event/game/player")
```

3. *Einschränken der Abfrage*. Um aus einer Liste von Elementen nur eine bestimmte Teilmenge zu extrahieren, können Bedingungen in Listenabfragen integriert werden. Nachfolgende Abfrage gibt nur den Spieler mit dem Namen „p1“ zurück:

```
XPathAPI.selectSingleNode(xmlObj, "/event/game/player[name=p1]")
```

4. *Zugriff auf Attribute*. Um auf Attribute eines Elements zugreifen zu können, muss dem Attributnamen ein „@“ vorangestellt werden. So kann beispielsweise über die folgende Abfrage der Name desjenigen Spielers erfragt werden, dem die Rolle des Großhändlers zugewiesen wurde:

```
XPathAPI.selectSingleNode(xmlObj,
        "/event/game/player[@level=Großhändler]/name").nodeValue()
```

6.2.3 Mehrsprachigkeit

Beim Frontend des Beergame besteht die Möglichkeit, die Sprache aller angezeigten Texte während des Betriebs ändern zu können. In der ersten Version sind dies Deutsch und Englisch, es soll jedoch zudem möglich sein, zu einem späteren Zeitpunkt weitere Übersetzungen hinzufügen zu können, ohne Anpassungen am Quellcode durchführen zu müssen. Um dies zu gewährleisten, sind mehrere Schritte durchzuführen:

Sämtliche in der Benutzeroberfläche vorkommenden Texte müssen aus dem Quellcode extrahiert und in externe Dateien ausgelagert werden. Im Quellcode darf ausschließlich auf diejenigen Positionen in diesen Dateien verwiesen werden, an denen sich der benötigte Textbaustein in der gewählten Sprache befindet. Die Dateien sollten eine einheitliche syntaktische Struktur aufweisen, um möglichst einfach Methoden zum Auffinden von Textbausteinen bereitstellen zu können. Auch hier bietet sich die Verwendung von XML an. Zum einen kann mit XML komfortabel eine einheitliche Datenstruktur definiert werden, zum anderen können bestehende Funktionen des Datenzugriffs weiterverwendet werden.

Die hierzu genutzte XML-Datenstruktur ist wenig komplex. Zumeist ist es ausreichend, zu jedem Textbaustein einen Bezeichner und den dazugehörigen Übersetzungstext anzugeben. Beispielsweise gibt das XML-Element

```
<label name="label_mail">Ihre Email-Adresse:</label>
```

die Beschriftung eines Eingebefelds an. Bei manchen Textbausteinen sind jedoch Zusatzinformationen notwendig. Etwa wird bei Spaltenköpfen von Tabellen zusätzlich die Breite der Spalte angegeben, damit die Spalten für jede Sprache die optimale Breite aufweisen und die Überschriften nicht abgeschnitten werden. Eine solche Spaltendefinition ist z. B.

```
<label name="column_incomingDelivery" width="95">Wareneingang</label>
```

Die Texte der Frontends für Spieler und Spielleiter befinden sich in verschiedenen Dateien. Zudem werden innerhalb der Dateien die genannten *label*-Elemente für jede Maske in einem *screen*-Element gruppiert. So können bei Bedarf nur die Texte einer bestimmten Oberfläche geladen werden.

In der Datei *languages.xml* befinden sich Informationen über alle derzeit verfügbaren Sprachen. Zu jeder Sprache wird ein Sprachkürzel angegeben, über welchen die einzelnen Sprachdateien identifiziert werden können. Beispielsweise lauten die Kürzel für Deutsch und Englisch „de“ bzw. „en“. Die deutschen Sprachdateien werden dann unter den Namen texts_player_de.xml bzw. texts_admin_de.xml gesucht.[60] Für weitere Übersetzungen sind die jeweiligen Kürzel in die Dateinamen einzusetzen.

Zudem wird für jede Sprache ein Verweis auf eine Grafik angegeben, welche in der Benutzeroberfläche ausgezeigt wird und als Schaltfläche zur Auswahl dieser Sprache dient. Für gewöhnlich handelt es sich dabei Icons entsprechender Landesflaggen. Bei der Initialisierung des Frontends wird die Datei *languages.xml* ausgelesen und die Auswahlschaltflächen für die zur Verfügung stehenden Sprachen angezeigt.

6.2.4 Diagramme

Zur grafischen Visualisierung des Verlaufs der Endkundenachfrage sowie bei der Spielbeobachtung werden Liniendiagramme angezeigt. Zu deren Umsetzung können die Zeichenfunktionen von Flash genutzt werden. Die Methoden zum Zeichnen der Diagramme sind in einem Movieclip enthalten, der frei auf der Benutzeroberfläche positioniert werden kann. Bevor das Diagramm gezeichnet werden kann, müssen in diesem Movieclip essenzielle Werte gesetzt worden sein. Diese sind:

- die Abmessungen des Diagramms,
- ein mehrdimensionales Array, die Koordinaten der zu zeichnenden Linien enthält,

[60] Gesucht wird ausschließlich im Verzeichnis /languages/.

- eine Information, ob eine Legende gezeichnet werden soll, und wenn ja,
- die Position der Legende sowie ein Array mit einem Legendentext für jede Linie,
- zudem ein Array, das für jede Linie die Information enthält, in welcher Art deren Punkte miteinander verbunden werden sollen.

Sind diese Informationen vorhanden, kann mittels der Methode *drawGraphs()* das Diagramm gezeichnet werden. Dies geschieht in mehreren Schritten: Zuerst wird das Koordinatenkreuz gezeichnet und die Einteilung der Achsenbeschriftungen ermittelt. Auf der waagerechten Achse werden beim Beergame ausschließlich Rundenzahlen und damit ganzzahlige Werte abgetragen. Neben der ersten und der letzten Runde wird der verbleibende Platz auf der Achse durch einen Algorithmus gleichmäßig aufgeteilt. Dabei können maximal fünf, jedoch mindestens zwei Teilstriche entstehen, je nach ganzzahliger Teilbarkeit der maximalen Rundenzahl.

Auf der vertikalen Achse müssen auch Werte mit Dezimalstellen abgetragen werden können. Dies ist beispielsweise bei Diagrammen über Kostenverläufe der Fall. Zudem muss berücksichtigt werden, dass für Beschriftungen auf der vertikalen Achse tendenziell weniger Platz zur Verfügung steht als auf der horizontalen Achse. Daher werden auf der vertikalen Achse neben dem in den Daten gefundenen Höchstwert noch die Hälfte bzw. die Quartile des Höchstwerts auf der Achse markiert.

Als Nächstes werden die Linien in das vorbereitete Koordinatensystem eingezeichnet. Jede Linie erhält dazu eine Farbe aus einer global im Voraus gefüllten Liste. Wie bereits erwähnt, muss für jede Linie definiert werden, auf welche Weise deren Punkte verbunden werden sollen. Zur Auswahl stehen zwei Möglichkeiten: zum einen eine direkte Verbindung zweier aufeinander folgender Punkte „Luftlinie“ durch genau eine Verbindungslinie. Die zweite Möglichkeit ist besteht darin, zuerst die horizontale Distanz mit einer Linie und danach senkrecht dazu die vertikale Distanz mit einer weiteren Linie zu überbrücken. Mit letzterer Variante können sprunghafte Anstiege, wie sie z. B. bei der Endkundennachfrage auftreten, visualisiert werden, wohingegen die erstere Variante das „klassische“ Liniendiagramm darstellt, wie es auch aus Tabellenkalkulationsprogrammen bekannt ist. Abb. 6.3 zeigt nochmals beide Varianten im Vergleich.

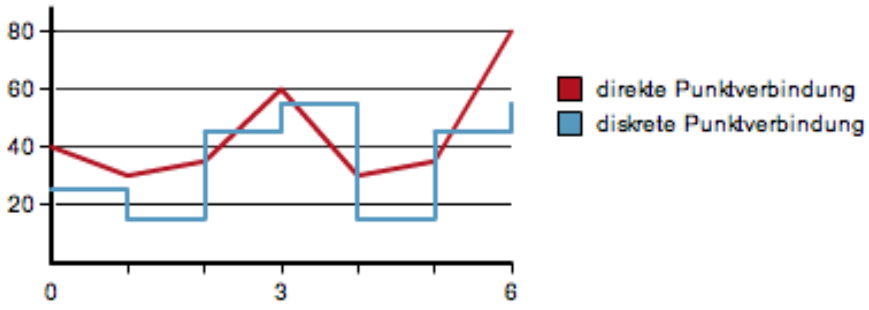


Abb. 6.3: Variationen des Liniendiagramms

Zuletzt wird, falls gewünscht, die Legende an die gewünschte Position gezeichnet. Links neben der Linienbezeichnung wird ein in der jeweiligen Farbe der Linie gefülltes Quadrat angezeigt. Jedes dieser Quadrate hat die Funktion einer Checkbox: Bei einem Klick auf eines der Quadrate wird das Diagramm neu gezeichnet, jedoch ohne die zuvor markierten Linien.

7 Mögliche Erweiterungen

Die Entwicklung von Software ist nie ein abgeschlossener Vorgang, sondern vielmehr ein Prozess, der über die gesamte Lebensdauer einer Software hinweg andauert.[61] Dabei verändern sich mit der Zeit auch die Anforderungen an die Software selbst, etwa an die Qualität aber vor allem auch an den Funktionsumfang der Applikation. Entwickler sollten daher schon zu Beginn der Entwicklung mögliche Erweiterungen antizipieren und deren spätere Umsetzung bestmöglich unterstützen.

Die vorliegende Implementierung des Beergame sieht vor allem vier Ansatzpunkte für spätere Erweiterungen vor, auf die im Folgenden näher eingegangen werden soll. Diese sind:

- die Anbindung einer neuen Benutzerschnittstelle,
- die Entwicklung externer Applikationen, die auf die Programmlogik des Beergame zugreifen,
- die Entwicklung externer Applikationen, die auf vom Beergame gespeicherte Daten zugreifen, und schließlich
- die Erweiterung der Programmlogik selbst.

7.1 Alternatives Frontend

Bei der Planung der vorliegenden Implementierung fiel die Entscheidung zur Benutzung einer bestimmten Frontend-Technologie, wie zuvor ausführlich dargelegt, zugunsten von Flash und gegen ein Frontend auf Basis von DHTML/Ajax aus. Es besteht jedoch durchaus die Möglichkeit, dass durch die weitere Entwicklung der letztgenannten Technik, vor allem im Bereich der Standardisierung und Konsolidierung der Ajax-Toolkits, die Entwicklung eines solchen Frontends für das Beergame ein sinnvoller Schritt sein könnte.

Bei der Planung wurde daher gesteigerter Wert darauf gelegt, die Programmlogik möglichst völlig vom Frontend zu trennen, und die Kommunikation zwischen Front- und Backend auf den Webstandard XML aufzubauen. Dadurch ist es weiterhin möglich, ein auf HTML basierendes Frontend zu bauen, das asynchron über die Ajax-Schnittstelle *XMLHttpRequest* mit dem Backend kommuniziert. Das Backend und die Controller des Beergame müssen für eine solche Erweiterung nicht angepasst werden. Es wird lediglich die Frontend-Sektion der Softwarearchitektur ausgetauscht.

[61] Vgl. Jacobson, Booch, Rumbaugh (1999), S.8 ff.

7.2 Alternativer Zugriff auf das Beergame Backend

Neben der Benutzerschnittstelle können auch externe Programme mit dem Backend kommunizieren. Ein nahe liegendes Beispiel für eine solche Applikation ist etwa ein Simulationsprogramm, welches automatisch und in kürzester Zeit eine große Anzahl von Spielen durchführen kann. Über diesen Weg lassen sich beispielsweise leicht die Auswirkungen verschiedener Anpassungen an der Konfiguration testen. Ein Programm dieser Art kann auf zwei Arten auf die Programmlogik des Beergame zugreifen:

1. *Über die Controller*: Analog zur Vorgehensweise des Frontends. Diese Methode ist unumgänglich, wenn sich das Beergame und die externe Applikation nicht auf derselben Maschine befinden und ein entfernter Zugriff über das Netzwerk erfolgen muss.

2. *Über die Beergame-API*: Java-Programme können die Beergame-Klassen als Bibliothek einbinden. Beergame-Aufrufe, können dann direkt in den Programmcode integriert werden. Das Exception-Handling des Beergame-Backend stellt dabei die Korrektheit der Eingaben und die Konsistenz der Aufrufe sicher.

7.3 Externer Zugriff auf gespeicherte Daten des Beergame

Schon während eines Spiels liegen die dabei anfallenden Daten in Form von XML-Dateien auf dem Server. Nach Beendigung kommt ein Excel-Spreadsheet dazu, welches die Ergebnisse aufbereitet. Die XML-Daten liegen in unverschlüsselter Form vor. Die Datenstruktur der XML-Dokumente ist unveränderlich und in einer Dokumenttypdefinition (DTD) festgelegt, die sich ebenfalls auf dem Server befindet.

Unter diesen Voraussetzungen ist es problemlos möglich, mit selbst entwickelten Applikationen, aber auch mit ausgewählter Standardsoftware auf diese Daten zuzugreifen. Ein Beispiel für eine selbst entwickelte Applikation ist etwa ein Programm, das die Ergebnisse eines Spiels oder auch einer ganzen Veranstaltung in eine Berichtsform bringt und eine PDF-Datei generiert. Auch andere Exporte sind über diesen Weg leicht umzusetzen.

Als Beispiel für den Zugriff von Standardsoftware auf Daten des Beergame seien hier Statistikpakete, wie etwa *GNU-R*, genannt.[62] Damit kann beispielsweise das Spielerverhalten mir empirischen Methoden statistisch analysiert werden, etwa um herauszufinden, wie hoch die Korrelation zwischen den Bestellungen der Spieler und der Endkundennachfrage ist, nachdem diese für den Spieler sichtbar gemacht wurde.

---

62 Zurm Import von XML-Daten in *R* siehe: http://cran.r-project.org/#doc, Abschnitt 1.3.

7.4 Erweiterung der Programmlogik

Eine Anpassung der im Backend implementierten Programmlogik wird beispielsweise dann notwendig, wenn die geplante Erweiterung eine Änderung der Datenstruktur zur Folge hat. Sollen etwa von einem Spieler mehr Informationen erfragt werden, als dies bisher der Fall ist, so müssten der Klasse *Player* weitere Attribute hinzugefügt und in die XML-Struktur dieser Klasse integriert werden. Weiterhin müssten die Controller-Aufrufe angepasst werden, sodass sie die zusätzlichen Informationen als Parameter anfordern. Bei Anpassungen dieser Art ist jedoch zu bedenken, dass Änderungen an der Datenstruktur eventuell zur Folge haben können, dass Daten, die vor der Änderung gespeichert wurden, nach der Änderung nicht mehr geladen werden können.

Ein weiteres Beispiel für eine mögliche Erweiterung des Backends besteht darin, den Computer als Mitspieler einsetzen zu können. Derzeit erfordert das Beergame in jedem Spiel die gleiche Anzahl an Spielern. Bei Gruppen unterschiedlicher Größe kann es jedoch vorkommen, dass bei einer Beergame-Veranstaltung in einem Spiel einer oder mehrere Plätze frei bleiben. In diesem Fall könnten die frei gebliebenen Plätze durch „Computergegner“ aufgefüllt werden, die ihre Bestellungen nach einer vorher vom Spielleiter festgelegten Bestellstrategie tätigen. Zu diesen Strategien zählt z. B. das Bestellen konstanter Mengen in jeder Runde, das Bestellen einer konstanten Menge bei Erreichung eines gewissen Mindestbestands oder das kontinuierliche Auffüllen des Lagers auf einen bestimmten Bestand.[63]

[63] Mehr zu Bestellstrategien siehe Becker (2004), S.295 ff.

8 Bedienung der Software

Nachdem mit Kapitel 6 die Implementierung der Beergame-Software abgeschlossen wurde, soll nun ein Rundgang durch die fertige Applikation die Ergebnisse dieser Arbeit aufzeigen. Die Vorgehensweise orientiert sich dabei an der praktischen Durchführung einer Beergame-Veranstaltung, angefangen bei der Konfiguration der Veranstaltung durch den Spielleiter, bis hin zur Durchführung der Spiele.

8.1 Benutzerschnittstelle des Spielleiters

8.1.1 Veranstaltung auswählen

Nach Aufruf des Programms wird dem Spielleiter zunächst eine Übersicht über alle derzeit auf diesem Server gestarteten Veranstaltungen (siehe Abb. 8.1) angezeigt. Für jede Veranstaltung werden der Veranstaltungstitel, ein kurzer Beschreibungstext sowie das Datum der Veranstaltung aufgelistet. Der Spielleiter hat nun die Möglichkeit, entweder eine dieser Veranstaltungen auszuwählen, eine neue Veranstaltung anzulegen, oder eine Veranstaltung von der Festplatte zu laden. Letztere Option kann beispielsweise dazu genutzt werden, sich die grafischen Auswertungen vergangener Veranstaltungen erneut anzusehen. Zudem können über diese Funktion Veranstaltungen, die wegen Zeitmangels nicht beendet werden konnten, wieder aufgegriffen und beendet werden.

Abb. 8.1: Übersicht der laufenden Veranstaltungen

8.1.2 Veranstaltung anlegen / Spielkonfiguration festlegen

Zum Anlegen einer Veranstaltung müssen zunächst einige beschreibende Informationen bezüglich der Veranstaltung und des Spielleiters angegeben werden. Danach wird der Spielleiter auf die Konfigurationsseite (siehe Abb. 8.2) weitergeleitet, um vor Beginn der Spiele die Spielkonfiguration festzulegen.

Abb. 8.2: Bildschirm zur Festlegung der Spielkonfiguration

Auf der Konfigurationsseite kann zunächst die Länge der Supply-Chain festgelegt werden. Hierzu befindet sich auf der oberen rechten Seite eine Liste, die bereits mit Standardwerten gefüllt wurde und vom Spielleiter angepasst werden kann. Die Länge der Supply-Chain sowie die Bezeichnung der Supply-Chain-Stufen sind variabel.

Im oberen linken Bereich befinden sich Steuerelemente zum Einstellen der Anfangswerte der Spieler zu Spielbeginn, also in der ersten Runde. Eingestellt werden können der Lagerbestand, der Wareneingang und der eingehende Kundenauftrag. Des Weiteren kann die Lieferdauer zwischen den Stufen der Supply-Chain, also der Shipping Delay, auf eine bestimmte Anzahl Runden festgelegt und mit einer festen Anzahl an sich unterwegs befindlichen Einheiten gefüllt werden.

Im mittleren Bereich kann zunächst links die Spieldauer auf eine bestimmte Anzahl Runden festgelegt werden. Sollte sich während des Spielverlaufs herausstellen, dass diese ma-

ximale Rundenzahl in der zur Verfügung stehenden Zeit nicht absolviert werden kann, so kann die maximale Rundenzahl auch während des Spiels angepasst werden. Über die Auswahlbox „Freigabe der nächsten Runde“ kann eingestellt werden, ob der Übergang von einer Spielrunde zur Nächsten automatisch vollzogen werden soll, sobald alle Spieler ihre Bestellung für entsprechende Runde abgegeben hat, oder ob der Spielleiter manuell für jedes der laufenden Spiele die nächste Runde freigeben muss. Letztere Variante erfordert zwar etwas mehr Interaktion seitens des Spielleiters, jedoch es lässt sich so leichter erreichen, dass die parallel, und generell unabhängig voneinander ablaufenden Spiele der Veranstaltung in ihrer Fortschrittsgeschwindigkeit nicht allzu sehr voneinander abweichen. Weiterhin kann der Spielleiter eine verzögerte Freigabe der Runde dazu nutzen, mit den Spielern mündlich oder über die Mailfunktion (s.u.) zu kommunizieren, ohne das die Spieler durch das laufende Spiel abgelenkt werden.

Rechts daneben kann mit den Auswahlboxen des Bereichs „Restriktionen für die Spieler“ eingestellt werden, welche Informationen den Spielern während des Spiels zur Verfügung gestellt werden. Angezeigt werden können der Shipping Delay, sowie die aktuelle Endkundennachfrage, die ansonsten nur für den Spieler in der Rolle des Einzelhändlers sichtbar ist. Das klassische Beergame sieht vor, dass die Spieler keinerlei Übersicht über den Verlauf der Endkundennachfrage oder die in den nächsten Runden eintreffenden Lieferungen haben. Es kann jedoch sinnvoll sein, den Spielern in einem weiteren Spiel zu verdeutlichen, dass durch höhere Informationsbereitstellung der Bullwhip-Effekt verringert werden kann.

Ein ähnlicher Effekt kann mit der Mailfunktion erreicht werden. Diese ist standardmäßig nur für die (bidirektionale) Kommunikation zwischen Spielern und Spielleiter aktiviert. Alternativ kann jedoch auch die Kommunikation zwischen den Spielern eines Spiels erlaubt werden, wodurch die Spieler ihre Unsicherheit bei der Bestellentscheidung reduzieren können.

Das Diagramm im unteren Bereich des Bildschirms zeigt den Verlauf der Endkundennachfrage. Während auf der horizontalen Achse die Spielrunden von der ersten Runde bis zur oben eingestellten maximalen Rundenzahl abgezeichnet sind, zeigt die vertikale Achse die Endkundennachfrage der jeweiligen Runde. Die Liste links neben der Grafik enthält eine zusammengefasste Darstellung: Es werden nicht alle Runden aufgeführt, sondern nur solche, in denen sich die Nachfrage im Vergleich zur Vorrunde ändert. Endkundennachfrage kann auf zwei Arten ins System eingegeben werden: zum einen durch manuelles Anpassen der Liste. Dies geschieht durch Eingabe von Wertepaaren der Form {ab RundeX, NachfrageY}. Alternativ kann die Endkundennachfrage auch über den Knopf „Zahlenfolge“ als Ganzes eingegeben werden. Akzeptiert werden Zeichenketten, in denen die Nachfragewer-

te, durch Kommata voneinander getrennt, hintereinander gereiht wurden. Über diesen Weg kann der Spielleiter Zahlenfolgen eingeben, die in externen Programmen generiert wurden, und denen beispielsweise Wahrscheinlichkeitsverteilungen zugrunde liegen.

Beim Anlegen einer neuen Veranstaltung werden die Felder der Einstellungsmaske stets mit Standardeinstellungen vorbefüllt. Es besteht jedoch die Möglichkeit, getätigte Änderungen über die Schaltflächen „Konfiguration laden“ bzw. „Konfiguration speichern“ auf dem Server zu sichern und bei späteren Veranstaltungen wieder zu verwenden.

8.1.3 Spieler zu Spielen zuordnen

Nachdem die Spielkonfiguration festgelegt wurde, wird der Spielleiter zur nächsten Maske weitergeleitet. In dieser Maske geht es darum, die bereits angemeldeten Spieler auf die Supply-Chain-Stufen der einzelnen Spiele aufzuteilen (siehe Abb. 8.3). Auf dem Bildschirm sind zwei Listen zu sehen. In der Linken werden die Spieler angezeigt, die bereits bei dieser Veranstaltung angemeldet sind, jedoch noch nicht zugeordnet wurden. Auf der rechten Seite befindet sich eine baumartig verschachtelte Liste. In dieser werden alle Spiele der Veranstaltung angezeigt. Die Spiele werden je nach Status in einen der drei Ordner auf der ersten Ebene des Baums einsortiert. Der erste Ordner enthält Spiele, bei denen noch eine oder mehrere Supply-Chain-Stufen nicht mit einem Spieler besetzt wurden. Der zweite Ordner enthält alle voll besetzten Spiele, die jedoch noch nicht gestartet wurden. Im dritten Ordner befinden sich letztlich alle laufenden Spiele der Veranstaltung. Klappt man einen der Ordner auf, werden darunter alle Spiele sichtbar. Wird ein Spiel aufgeklappt, erscheinen die Supply-Chain-Stufen mit den Namen der zugeordneten Spieler.

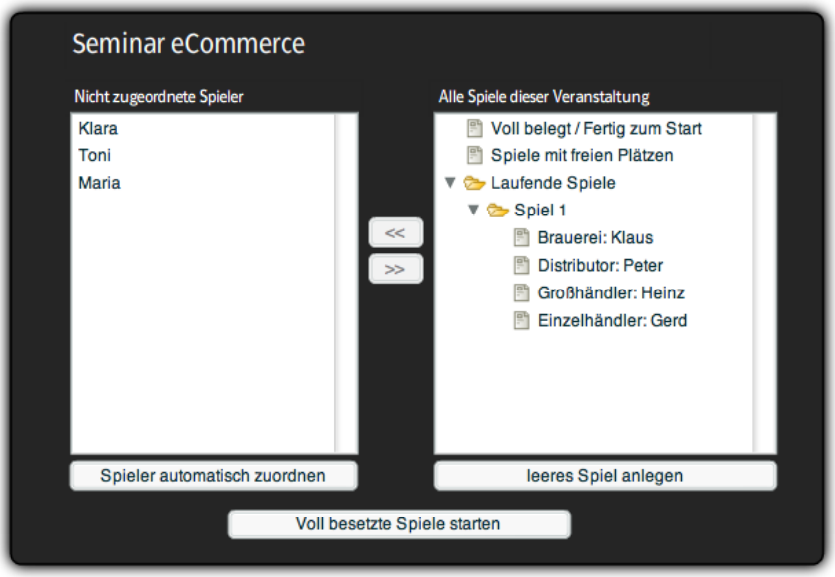


Abb. 8.3: Bildschirm zur Zuordnung von Spielern zu Spielen

Die Zuordnung der Spieler kann auf zwei Weisen erfolgen. Bei Wahl der Funktion „Spieler automatisch zuordnen“ füllt das System selbst freie Plätze in den Spielen mit wartenden Spielern auf und legt im Bedarfsfall neue Spiele an. Möchte der Spielleiter explizit bestimmte Spieler gruppieren, so kann er manuell leere Spiele anlegen und diese im Anschluss befüllen. Dazu müssen zunächst ein Spieler aus der Liste auf der linken Seite sowie eine Position innerhalb eines Spiels in der rechten Liste markiert werden. Per Klick auf den Knopf mit den rechtsgerichteten Pfeilen („>>“) wird die Zuordnung durchgeführt. Über die Schaltfläche mit den linksgerichteten Pfeilen („<<“) können einzelne markierte Positionen in Spielen wieder zur Neubesetzung freigegeben werden. Die betroffenen Spieler erscheinen nach der Freigabe wieder in der Liste auf der linken Seite.

8.1.4 Alle Spiele beobachten

Nach dem Zuordnen der Spieler und dem Starten der Spiele können nun die laufenden Spiele beobachtet werden. Hierzu gibt es eine Übersichtstabelle, in der jedes Spiel eine Zeile einnimmt (siehe Abb. 8.4). In jeder Zeile finden sich Informationen über die aktuelle Runde des Spiels, den momentanen Status der Spielrunde sowie eine Auswahl wichtiger Werte der Spieler. Der Status einer Spielrunde kann drei verschiedene Werte annehmen:

- „läuft“, wenn noch nicht alle Spieler ihre Bestellungen abgegeben haben,
- „wartet auf Freigabe“, wenn alle Bestellungen abgegeben wurden, in den Einstellungen jedoch manuelle Rundenfreigabe eingestellt wurde

- sowie „Spiel beendet“.

Für jede Supply-Chain-Stufe werden in einer Spalte aktuelle Rundendaten des dazugehörigen Spielers zusammengefasst angezeigt. Diese sind der aktuelle Lagerbestand (bei Lieferrückstand evtl. negativ), die Bestellung der letzten Runde sowie die kumulierten Gesamtkosten. Ein vorangestelltes „X“ zeigt an, ob der Spieler schon seine Bestellung abgegeben hat.

Abb. 8.4: Bildausschnitt der Spielübersichtsmaske

Diese Seite stellt für den Spielleiter während des Spiels eine wichtige Informationsquelle dar. Hier können der Spielverlauf verfolgt, die Werte verschiedener Spiele miteinander verglichen und bei Bedarf per Knopfdruck nächste Runden freigegeben werden. Des Weiteren kann über die Schaltfläche „Spieldetails anzeigen“ oder auch per Doppelklick auf eine Tabellenzeile in eine Detailansicht gewechselt werden, in der rundengenaue Daten des jeweiligen Spiels in tabellarischer und grafischer Form angezeigt werden.

Nicht vergessen werden darf, dass der Spielleiter jederzeit die Möglichkeit hat, über die Navigation an oberen Bildrand zu den zuvor beschriebenen Seiten „Konfiguration ändern“ und „Spieler zuordnen“ zurückzuspringen, um neu hinzugekommenen Spielern ein Spiel einzurichten oder um während des Spielbetriebs die Spiellänge oder die Endkundennachfrage anzupassen.

8.1.5 Detailansicht eines Spiels

Während in der zuvor beschriebenen Spielübersicht nur Informationen über den aktuellen Stand der Spiele angezeigt wurden, kann in der Detailansicht der genaue Verlauf eines Spiels nachvollzogen werden (siehe Abb. 8.5). Die Tabelle auf der linken Seite enthält ähnliche Spalten wie die Spielübersichtstabelle, d. h. wiederum eine Auswahl aus den Rundendaten der Spieler. Jede Zeile enthält die Ausprägungen dieser Werte in einer bestimmten Runde. Der Spielfortschritt sowie der Status der aktuellen Runde befinden sich

rechts neben der Tabelle. Auch hier taucht der Benutzerfreundlichkeit halber der Knopf zum Freigeben der nächsten Runde wieder auf.

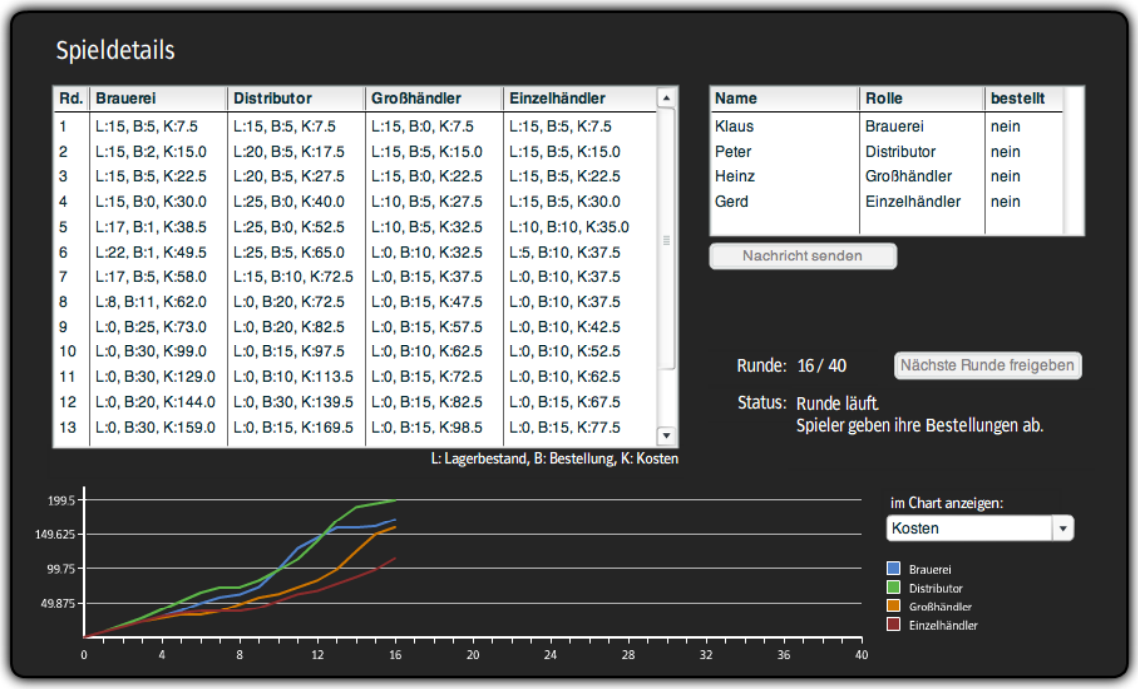

Spieldetails
Rd. Brauerei Distributor Großhändler Einzelhändler
1 L:15, B:5, K:7.5 L:15, B:5, K:7.5 L:15, B:0, K:7.5 L:15, B:5, K:7.5
2 L:15, B:2, K:15.0 L:20, B:5, K:17.5 L:15, B:5, K:15.0 L:15, B:5, K:15.0
3 L:15, B:5, K:22.5 L:20, B:5, K:27.5 L:15, B:0, K:22.5 L:15, B:5, K:22.5
4 L:15, B:0, K:30.0 L:25, B:0, K:40.0 L:10, B:5, K:27.5 L:15, B:5, K:30.0
5 L:17, B:1, K:38.5 L:25, B:0, K:52.5 L:10, B:5, K:32.5 L:10, B:10, K:35.0
6 L:22, B:1, K:49.5 L:25, B:5, K:65.0 L:0, B:10, K:32.5 L:5, B:10, K:37.5
7 L:17, B:5, K:58.0 L:15, B:10, K:72.5 L:0, B:15, K:37.5 L:0, B:10, K:37.5
8 L:8, B:11, K:62.0 L:0, B:20, K:72.5 L:0, B:15, K:47.5 L:0, B:10, K:37.5
9 L:0, B:25, K:73.0 L:0, B:20, K:82.5 L:0, B:15, K:57.5 L:0, B:10, K:42.5
10 L:0, B:30, K:99.0 L:0, B:15, K:97.5 L:0, B:10, K:62.5 L:0, B:10, K:52.5
11 L:0, B:30, K:129.0 L:0, B:10, K:113.5 L:0, B:15, K:72.5 L:0, B:10, K:62.5
12 L:0, B:20, K:144.0 L:0, B:30, K:139.5 L:0, B:15, K:82.5 L:0, B:15, K:67.5
13 L:0, B:30, K:159.0 L:0, B:15, K:169.5 L:0, B:15, K:98.5 L:0, B:15, K:77.5
L: Lagerbestand, B: Bestellung, K: Kosten
Name Rolle bestellt
Klaus Brauerei nein
Peter Distributor nein
Heinz Großhändler nein
Gerd Einzelhändler nein
Nachricht senden
Runde: 16 / 40
Nächste Runde freigeben
Status: Runde läuft.
Spieler geben ihre Bestellungen ab.
199.5
149.625
99.75
49.875
0 4 8 12 16 20 24 28 32 36 40
im Chart anzeigen:
Kosten
Brauerei
Distributor
Großhändler
Einzelhändler

Abb. 8.5: Bildschirm zur detaillierten Beobachtung eines Spiels

Die Tabelle in der rechten oberen Ecke zeigt Informationen über die Spieler dieses Spiels an, deren Name, die zugewiesene Rolle, sowie ein Indikator, ob der Spieler in der aktuellen Runde schon seine Bestellung abgegeben hat.

An unteren Bildrand befindet sich ein Liniendiagramm. In diesem Diagramm kann sich der Spielleiter die Entwicklung eines Wertes im Rundenverlauf grafisch ansehen. Zur Verfügung stehen die Werte „Lagerbestand“, „Lieferrückstand“, „Bestellung“ und „Gesamtkosten“. Das Diagramm enthält die Werte aller Spieler, wobei jeder Spieler durch eine Linie anderer Farbe repräsentiert wird. Per Klick auf die Quadrate neben den Legendentexten können einzelne Linien ein- bzw. ausgeblendet werden.

8.1.6 Nachrichten senden / empfangen

Während einer Veranstaltung hat der Spielleiter jederzeit die Möglichkeit, einem oder mehreren Spielern eine Textnachricht zukommen zu lassen. Dies kann über zwei Wege geschehen: Zum einen befindet sich in der Spielübersicht unter der Tabelle eine Schaltfläche „Nachricht senden“. Hierüber kann eine Nachricht an alle Spieler der in der Tabelle

markierten Spiele gesendet werden. Die andere Möglichkeit ist, in der Detailansicht eines Spiels einen oder mehrere Spieler in der Spielertabelle auszuwählen und wiederum den Knopf „Nachricht senden“ zu drücken.

In beiden Fällen öffnet sich anschließend eine Maske zum Senden von Nachrichten (siehe Abb. 8.6 links). In dieser Maske befinden sich in einer Liste auf der linken Seite alle möglichen Empfänger, wobei die zuvor getroffene Auswahl des Spielleiters in dieser Liste bereits berücksichtigt ist. Der Nachrichtentext kann in das Textfeld auf der rechten Seite eingegeben werden.

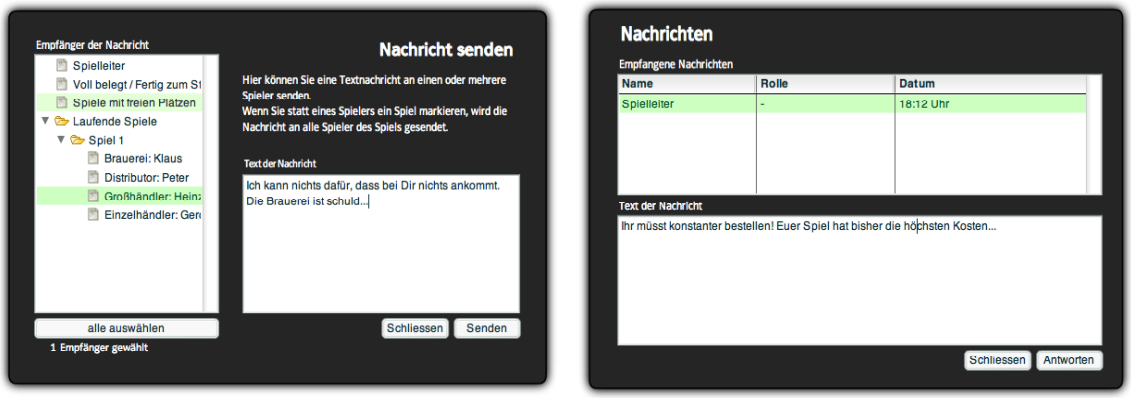


Abb. 8.6: Fenster zum Senden und Empfangen von Nachrichten

Da der Spielleiter auch Nachrichten von Spielern empfangen kann, steht eine Posteingangsmaske zur Verfügung, die über das Brief-Icon am oberen Bildrand erreicht werden kann (siehe Abb. 8.6 rechts). Trifft eine neue Nachricht ein, so erscheint neben dem Icon ein Hinweistext, solange die Nachricht nicht gelesen wurde. Der Posteingang besteht aus einer Tabelle, in der Informationen über alle eingegangenen Nachrichten aufgelistet werden, sowie einem Textfeld, welches den Nachrichtentext der ausgewählten Nachricht anzeigt. Ist eine Nachricht markiert, kann über die Schaltfläche „Antworten“ wiederum das Sendenfenster geöffnet werden.

8.2 Benutzerschnittstelle des Spielers

8.2.1 Veranstaltung betreten

Zu Beginn muss ein Spieler zunächst eine Veranstaltung betreten (siehe Abb. 8.7). Alle derzeit laufenden Veranstaltungen sind im entsprechenden Dropdown-Feld aufgelistet. Zum Betreten einer Veranstaltung werden vom Spieler dessen Name und Email-Adresse erfragt.

Abb. 8.7: Anmeldebildschirm

Nach dem Betreten wird dem ein Wartebildschirm angezeigt. Erst wenn das Spiel beginnt, verschwindet dieser Bildschirm wieder und der Spieler wird automatisch zur Spielansicht weitergeleitet. Über das Dropdown-Feld in der rechten oberen Ecke des Bildschirms kann der Spieler anschließend zwischen den beiden alternativen Spielansichten, der Tabellen- und der Animationsansicht hin und her wechseln.

8.2.2 Spielansicht I (Tabelle)

In der Tabellenansicht hat der Spieler den kompletten Spielverlauf im Blick (siehe Abb. 8.8). Die Tabelle enthält für jede Spielrunde eine Zeile. Aus dieser Zeile kann herausgelesen werden, in welchen Höhen sich die Auftrags- und Lagerbestände sowie die Gesamtkosten in dieser Runde entwickelt haben, abhängig von den eingehenden und ausgehenden Bestellungen und Lieferungen. Je nach Spielkonfiguration kann die Tabelle in zusätzlichen Spalten die Endkundennachfrage sowie den Inhalt des Shipping Delay enthalten. Um den Ablauf der Spielrunde nachzuvollziehen, können die Spalten von links nach rechts gelesen werden: Zunächst wird der Wareneingang ins Lager bewegt. Wareneingang und bisheriger Lagerbestand ergeben addiert den verfügbaren Bestand, also die Menge an Waren, die theoretisch ausgeliefert werden könnte. Danach wird die eingehende Bestellung entgegengenommen und zu den bereits vorhandenen offenen Bestellungen addiert. Die offenen Be-

stellungen werden so vollständig wie möglich gedeckt, wonach schließlich die neuen Bestände (Lager und offene Bestellungen) notiert werden können. Aus diesen Beständen ergeben sich die Kosten der Runde, welche zu den bisherigen Gesamtkosten addiert werden.

Woche	Wareneingang	Verfügbar	Kundenauftrag	Warenausgang	Offen	Lagerbestand	Kosten	Meine Bestellung
1	5	20	5	5	0	15	7.5	0
2	5	20	5	5	0	15	15.0	5
3	5	20	5	5	0	15	22.5	0
4	0	15	5	5	0	10	27.5	5
5	5	15	5	5	0	10	32.5	5
6	0	10	10	10	0	0	32.5	10
7	5	5	10	5	5	0	37.5	15
8	5	5	10	5	10	0	47.5	15
9	10	10	10	10	10	0	57.5	15

Abb. 8.8: Bildausschnitt der Tabellenansicht

Nach jedem Start einer neuen Runde wird eine weitere Zeile in die Tabelle eingefügt. Bis auf die Bestellung des Spielers ist die Zeile bereits vollständig ausgefüllt. Der Spieler kann nun mit Blick auf die vorliegenden Daten seine Bestellentscheidung treffen und gibt den gewünschten Bestellwert in das Eingabefeld am rechten unteren Rand der Tabelle ein. Nachdem der Spieler seine Bestellung eingegeben und bestätigt hat, erscheint ein Popup-Fenster auf dem Bildschirm, welches ihn darauf hinweist, dass die getroffene Bestellentscheidung korrigiert werden kann, solange sich das Spiel noch in der entsprechenden Runde befindet. Wird die nächste Runde vom System freigegeben, verschwindet das Popup wieder, und die Tabelle wird mit den Werten der neuen Runde aktualisiert.

8.2.3 Spielansicht II (Animation)

Die Tabellenansicht bietet einen sehr guten Überblick über die momentane Spielsituation, jedoch ist sie eher für erfahrene Spieler gedacht, die das Beergame nicht zum ersten Mal spielen und somit die Material- und Informationsflüsse innerhalb einer Supply-Chain bereits verinnerlicht haben. Beim Brettspiel haben die Spieler eine abstrahiere Supply-Chain als Spielbrett vor sich liegen, auf dem Waren und Bestellungen nach einem genau bestimmten Ablauf bewegt werden müssen. So bekommt der Spieler schnell ein Gefühl für das Spiel. In der vorliegenden Software gibt es daher eine weitere Spielansicht, welche die Herkunft, bzw. die Entstehung der in der Tabellenansicht zu sehenden Daten grafisch darstellt (siehe Abb. 8.9).

Abb. 8.9: Animationsansicht

In der Animationsansicht sieht der Spieler hauptsächlich einen Ausschnitt der Supply-Chain: sich selbst sowie die beiden umliegenden Stufen. An welcher Position in der Supply-Chain sich dieser Ausschnitt genau befindet, wird in einer stilisierten Abbildung der gesamten Supply-Chain am oberen Bildrand angezeigt.

Jede Stufe der Supply-Chain wird durch eine Art Spielblatt symbolisiert, das in zwei Bereiche unterteilt ist. Im unteren Bereich befinden sich die aktuellen Bestände für offene Bestellungen und Lager sowie die laufenden Gesamtkosten. Im oberen Bereich sieht man die einzelnen Phasen einer Spielrunde. Diese werden in jeder Runde wie eine Checkliste abgearbeitet. Die Spielblätter der umliegenden Stufen sind nur angedeutet, es sind keine Daten sichtbar. Sie werden jedoch angezeigt, um die Material- und Informationsflüsse während der Runde darstellen zu können. An den Rändern der Supply-Chain werden anstelle der benachbarten Spielblätter die Fabrik bzw. die Endkunden durch entsprechende Grafiken symbolisiert. Die Material- und Informationsflüsse werden durch Icons dargestellt, die sich zwischen den Spielblättern der Stufen bewegen. Zwischen den Stufen befinden sich, analog zum Brettspiel, Felder für den Shipping Delay, sowie ein weiteres Feld „Postweg“, welches versendete Bestellungen aufnimmt und an die nächste Stufe weiterleitet. Ähnliche Felder befinden sich auch zwischen Brauerei und Fabrik. Diese nehmen den

Produktionsauftrag der Brauerei (Production Request) sowie die Produktionsdauer (Production Delay) auf.

Nach dem Start einer neuen Runde wird nun eine Animation abgespielt, welche die Phasen der Spielrunde, welche im oberen Bereich des Spielblatts aufgelistet sind, nacheinander durchführt. Die Animation jeder Phase ist etwa zehn Sekunden lang, währenddessen wird die Bezeichnung der aktuellen Phase auf dem Spielblatt rot hervorgehoben.

Zunächst kommt eine neue Lieferung an. Der LKW des obersten Shipping Delay Feldes (Production Delay im Falle der Brauerei) bewegt sich nach rechts auf das Spielblatt, der untere LKW rückt auf das freigewordene Feld. Der angekommene LKW wird entladen, das LKW-Icon ändert sich in das einer offenen Kiste. In derselben Zeile des Spielblatts erscheint die Menge der angekommenen Waren, die sogleich ins Lager überführt werden. Kommen etwa 5 Einheiten an, erscheint neben dem Betrag des Lagerbestands zunächst die Information „(+5)“, danach ändert sich der Lagerbestand dementsprechend.

In der zweiten Phase wird die neue Bestellung entgegengenommen. Der Auftrag wird von der Post zugestellt, der Brief wandert vom Feld Postweg in die Zeile der zweiten Phase. Danach wird der Brief geöffnet, woraufhin die Bestellmenge erscheint. Diese Menge wird nun zum Wert der offenen Bestellungen addiert, die Animation hierzu verläuft wie schon das Buchen des neuen Lagerbestands zuvor. Die Fabrik nimmt in dieser Phase den Auftrag vom Feld *Production Request* entgegen, die gewünschte Produktionsmenge wird dann auf dem unteren der *Production Delay* Felder platziert.

In der dritten Phase werden die offenen Bestellungen ausgeliefert. Dazu erscheint in der entsprechenden Zeile eine offene Kiste, die nun befüllt werden muss. Die Liefermenge ergibt sich aus dem kleineren der Werte „Lagerbestand“ und „offene Bestellungen“. Diese Menge wird nun aus beiden Beständen ausgebucht und erscheint danach neben der offenen Kiste. Die Kiste wird zunächst geschlossen, danach ändert sich Icon der Kiste in einen LKW, der daraufhin auf das untere Shipping Delay Feld fährt. Im Falle des Einzelhändlers werden die Waren direkt an die Endkunden geliefert.

In der letzten Phase muss der Spieler seine Bestellung abgeben. Dazu erscheinen in der vierten Zeile ein Eingabefeld sowie ein offener Briefumschlag. Bestätigt der Spieler die Bestellung, schließt sich der Umschlag und bewegt sich auf das Feld „Postweg“ links neben dem Spielblatt des Spielers. Wie auch in der Tabellenansicht erscheint nun das Popup mit dem Hinweis auf die Möglichkeit zur Korrektur der Bestellung.

Die genannten Animationen werden immer für alle drei sichtbaren Supply-Chain-Stufen gleichzeitig ausgeführt. Liefert der Spieler beispielsweise in Stufe 3 Waren aus, so sieht er

gleichzeitig, dass auch der Spieler auf der linken Seite Waren an ihn liefert. Gibt der Spieler eine Bestellung ab, so kommt auch im anderen Postweg auf der linken Seite eine Bestellung an, usw. So kann der Spieler bereits nach wenigen Runden verstehen, welche Informationen bzw. Waren sich wann wo befinden.

Bei Testspielen während der Entwicklung zeigte sich, dass Spieler zumeist in den ersten Runden die Animationsansicht bevorzugen. Fühlen sie sich dann mit dem Spielablauf hinreichend vertraut, wechseln sie oft auf für den Rest des Spiels zur Tabellenansicht.

8.2.4 Nachrichten senden / empfangen

Wie bereits erwähnt, können die Spieler während des Spiels Nachrichten vom Spielleiter empfangen. Inwieweit den Spielern auch das Senden von Nachrichten möglich ist, hängt von der Konfiguration der Veranstaltung ab: Grundsätzlich ist es immer gestattet, Nachrichten an den Spielleiter zu senden. Der Nachrichtenaustausch mit Mitspielern muss dagegen explizit vom Spielleiter erlaubt werden.

Grundsätzlich sind die Benutzeroberflächen zum Senden und Empfangen von Nachrichten sowohl beim Spielleiter als auch bei den Spielern gleich aufgebaut, weshalb hier auf die Erläuterungen in Kapitel 8.1.6 verwiesen sei. Das Fenster zum Senden von Nachrichten kann im Falle der Spieler über eine zusätzliche Schaltfläche in der Posteingangsmaske erreicht werden.

9 Fazit nach der Entwicklung

Die ersten Tests mit der fertigen Software konnten als durchaus viel versprechend bewertet werden. Es zeigte sich, dass innerhalb kürzester Zeit eine Beergame-Veranstaltung gestartet und durchgeführt werden konnte. Dabei spielte die Anzahl der Spieler eine wie erhofft geringe Rolle. Nachdem den Spielern zu Beginn einige einleitende Worte zum Thema Bullwhip-Effekt und dem Ziel des Beergame mit auf den Weg gegeben wurden, waren den Meisten die Funktionsweise der Software und der Ablauf des Spiels schnell klar. Wie eine nachträgliche Befragung ergab, war den Spielern dazu vor allem die Animationsansicht eine große Hilfe, da die Animation die durch den Wegfall von Spielbrett und Spielsteinen fehlenden haptischen und visuellen Elemente des Spiels adäquat kompensieren konnte.

Die technischen Vorbereitungen zu Beginn einer Veranstaltung nahmen bei den durchgeführten Tests meist nur geringe Zeit in Anspruch. Es wurden zwei Szenarien durchgespielt. Bei dem ersten Szenario befand sich der Webserver mit der Software bereits auf dem Laptop des Spielleiters. Da die Laptops der Spieler teilweise mit unterschiedlichen Betriebssystemen ausgerüstet sein können, sollte der Spielleiter in diesem Fall jedoch über entsprechendes technisches Hintergrundwissen verfügen, um die Laptops über ein ad-hoc Drahtlosnetzwerk zu verbinden. Das zweite Szenario stellte den Betrieb der Software in einem fest installierten Rechnerverbund dar, wobei sich der Webserver auf einem vorkonfigurierten USB-Stick befand. Diese Variante stellte weitaus weniger Konfigurationsaufwand für den Spielleiter dar, da die Rechner sind in diesem Fall bereits vernetzt waren. Nach dem Starten des Servers direkt vom Stick musste den Spielern nur noch die URL des Servers mitgeteilt werden.

Vergleicht man die vorliegende Software mit den eingangs angeführten Brettspielvarianten, wird man feststellen, dass die wesentlichen zur Durchführung des Beergame notwendigen Funktionen umgesetzt wurden. Wird mit der Software das Beergame in Standardkonfiguration gespielt, werden annähernd dieselben Ergebnisse erzielt wie durch das Brettspiel. Darüber hinaus bietet die Software allerdings durch die vielfältigen Konfigurationsmöglichkeiten den Vorteil, gezielter die Ursachen und Wirkungen des Bullwhip-Effekts erarbeiten zu können.

In Kapitel 7 wurde deutlich, dass es neben den Grundfunktionen des Beergame noch zahlreiche weitere Möglichkeiten gibt, die Software oder die aus ihrer Benutzung hervorgehende Datenmenge zu nutzen. Die vorliegende Umsetzung stellt hierfür eine angemessene Grundlage dar. Architekturen und Datenformate wurden so offen wie möglich gewählt, um eine größtmögliche Anzahl an Schnittstellen für weitere Entwicklungen bereitzustellen, denen sich weitere Arbeiten widmen könnten.

Literaturverzeichnis

Alicke, K.: Planung und Betrieb von Logistiknetzwerken. Unternehmensübergreifendes Supply Chain Management. 2. Aufl. Berlin. 2005.

Allaire, J.: Macromedia Whitepaper: *Flash MX - A Next Generation Rich Client,* March 2002.

Balzert, H.: Lehrbuch der Software-Technik. Bd.1. Software-Entwicklung, Heidelberg 1996.

Becker, J.; Schütte, R.: Handelsinformationssysteme. 2. Aufl., Frankfurt/Main 2004.

Bergsten, H.: Java Server Pages. Peking u.a. 2002.

Chopra, S.; Meindl, P.: Supply-Chain Management. Upper Saddle River, NJ 2001.

Forrester, J.W.: Industrial Dynamics. A major breakthrough for decision makers. In: Harvard Business Review, 36 (1957) 4, S. 37–66.

Forrester, J.W.: Industrial Dynamics. Cambride, MA 1961.

Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J.: Entwurfsmuster. Elemente wieder verwendbarer objektorientierter Software. München u.a. 2004.

Günthner, W.A.: Wissenschaft in der Logistik – 14. Deutscher Materialfluss-Kongress „Intralogistik – Innovation und Praxis". In: VDI-Berichte, Nr. 1882. Düsseldorf 2005.

Jacobson, I.; Booch, G.; Rumbaugh, J.: The Unified Software Development Process. Reading/MA u.a. 1999.

Jacobson, I.; Christerson, M.; Jonsson, P.; Övergaard, G.: Object-Oriented Software Engineering. A Use Case Driven Approach. Wokingham u.a. 1992.

Klettke, M.; Meyer, H.: XML & Datenbanken. Konzepte, Sprachen, Systeme. Heidelberg 2003.

Lee, H. L.; Padmanabhan, V.; Whang, S.: The Bullwhip Effect in Supply-Chains. Sloan Management Review, 38 (1997) 3, S. 93-102.

Lehner, W.; Schöning, H.: XQuery. Grundlagen und fortgeschrittene Methoden. Heidelberg 2004.

Louis, D.; Müller, P.: Java 5. Praxis der objektorientierten Programmierung. München 2005.

Macromedia Developer Network: Flash 8. ActionScript 2.0 Language Reference. 2005a. http://download.macromedia.com/pub/documentation/en/flash/fl8/fl8_as2lr.pdf

Macromedia Developer Network: Flash 8. XPath Class API. Macromedia 2005b. http://download.macromedia.com/pub/documentation/en/flash/fl8/XpathAPI.pdf

O'Reilly, T.: What is Web 2.0. Design Patterns and Business Models for the Next Generation of Software, http://www.oreillynet.com/lpt/a/6228, 09/30/2005

Ossimitz, G.: Endbericht zum Projekt „Simulation von Supply-Chain-Management Systemen". Klagenfurt 2002. http://www.uni-klu.ac.at/~gossimit/pap/bg_endbericht.pdf

Peterson, L. L.; Davie, B. S.: Computer Networks. A System Approach. Edition 3. San Francisco 2003.

Radke, H.-D.; Radke, J.: Wireless LAN. Drahtlos glücklich. München 2005.

Riley, D.: The Object of Java. Introduction to Programming using Software Engineering Principles. Boston u.a., 2002.

Schönsleben, P.: Integrales Logistikmanagement. Operations und Supply-Chain-Management in umfassenden Wertschöpfungsnetzwerken. 5. Aufl., Berlin u.a. 2007.

Schulte, C.: Logistik. Wege zur Optimierung der Supply-Chain. 4. Aufl., München 2005.

Simchi-Levi, D.; Kaminsky, P.; Simchi-Levi E.: Designing & Managing the Supply-Chain. 2. Aufl., Boston, MA u.a. 2003.

Sterman, J. D.: Teaching Takes Off. Flight Simulators for Management Education. The Beer Game. In: OR/MS Today, Oktober 1992, S. 40-44, auch unter http://web.mit.edu/jsterman/www/SDG/beergame.html

Tanenbaum, A.; Steen, M. v.: Verteilte Systeme. Grundlagen und Paradigmen. München u.a., 2003.

Vossen, G., Hagemann, S.: Unleashing Web 2.0. From Concepts to Creativity. Amsterdam u.a. 2007.

Warburton, R.: An Analytical Investigation of the Bullwhip Effect. In: Production and Operations Management, 13 (2004) 2, S. 150–160.

Weber, J.: Components of the Bullwhip Effect. 2001. http://www.stud.fernuni-hagen.de/q5153735/research/bullwhiptop.htm. Abrufdatum: 2005-08-23.

Wöhe, G.: Einführung in die Allgemeine Betriebswirtschaftslehre. 20. Auflage. München 2000.

Anhang

A XML Datenstruktur

A.a Event

Definition

```
<!ELEMENT event (eventKey, title, description, date, eventFolder, prefer-
          ences, pendingPlayers, games, admin)>
<!ELEMENT eventKey (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT eventFolder (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT pendingPlayers (player)*>
<!ELEMENT games (game)*>
```

Beispiel

```
<event>
     <eventKey>12345</eventKey>
     <date>12345</date>
     <title>
          Titel der Veranstaltung
     </title>
     <description>
          Beschreibung der Veranstaltung
     </description>
     <eventFolder>
          Speicherort auf der Festplatte
     </eventFolder>
     <preferences>
          ...preferences...
     </preferences>
     <pendingPlayers>
          ...players...
     </pendingPlayers>
     <games>
          ...games...
     </games>
     <admin>
          ...admin...
     </admin>
</event>
```

A.b Admin

Definition

```
<!ELEMENT admin (name, email)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT email (#PCDATA)>
```

Beispiel

```
<admin>
     <name>Name des Spielleiters</name>
     <email>Dessen Email Adresse</email>
</admin>
```

A.c Game

Definition

```
<!ELEMENT game (gameKey, gameState, currentRound, hasEnoughPlayers,
          sc_levels)>

<!ELEMENT gameKey (#PCDATA)>
<!ELEMENT gameState (#PCDATA)>
<!ELEMENT currentRound (#PCDATA)>
<!ELEMENT hasEnoughPlayers (#PCDATA)>

<!ELEMENT sc_levels (level)*>
<!ATTLIST roundData    id    CDATA #REQUIRED
                       label CDATA #REQUIRED>
<!ELEMENT level (player)>
```

Beispiel

```
<game>
     <gameKey>12345</gameKey>
     <gameState>2000</gameState> (= running)
     <currentRound>2</currentRound>
     <hasEnoughPlayers>true</hasEnoughPlayers>
     <sc_levels>
          <level id="Brauerei">
               ...player...
          </level>
     </sc_levels>
</game>
```

A.d Player

Definition

```
<!ELEMENT player (playerKey, scLevel, name, email, madeTurn, assignedTo-
          Game, rounds)>
<!ELEMENT playerKey (#PCDATA)>
<!ELEMENT scLevel (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT madeTurn (#PCDATA)>
<!ELEMENT assignedToGame (#PCDATA)>
<!ELEMENT rounds (roundData)+>
```

Beispiel

```
<player>
     <playerKey>12345</playerKey>
     <name>Florian Christ</name>
     <email>florian.christ@gmx.net</email>
     <madeTurn>true</madeTurn>
     <assignedToGame>true</assignedToGame>
     <rounds>
          ...roundData...
     </rounds>
</player>
```

A.e Message

Definition

```
<!ELEMENT message (content, date, sender)>
<!ELEMENT content (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT sender (player | admin)>
```

Beispiel

```
<message>
     <date>123123123</date> (timestamp)
     <content>Dies ist eine Nachricht!</content>
     <sender>
          ...player...  (oder „admin“)
     </sender>
</message>
```

A.f RoundData

Definition

```
<!ELEMENT roundData (shippingIncomingDeliveries, stockBeforeRound, backo-
          derBeforeRound, incomingDelivery, incomingOrder, outgoingDeliv-
          ery, outgoingOrder, stockAfterRound, backorderAfterRound,
          roundCosts)>

<!ATTLIST roundData   round       CDATA #REQUIRED>

<!ELEMENT shippingIncomingDeliveries (delivery)+>
<!ELEMENT delivery                   (#PCDATA)>
<!ATTLIST delivery    id             CDATA #REQUIRED>

<!ELEMENT stockBeforeRound           (#PCDATA)>
<!ELEMENT backorderBeforeRound       (#PCDATA)>
<!ELEMENT incomingDelivery           (#PCDATA)>
<!ELEMENT incomingOrder              (#PCDATA)>
<!ELEMENT outgoingDelivery           (#PCDATA)>
<!ELEMENT outgoingOrder              (#PCDATA)>
<!ELEMENT backorderAfterRound        (#PCDATA)>
<!ELEMENT stockAfterRound            (#PCDATA)>
<!ELEMENT roundCosts                 (#PCDATA)>
```

Beispiel

```
<roundData round="2">
     <shippingIncomingDeliveries>
          <delivery id="0">5</delivery>
          <delivery id="1">7</delivery>
     </shippingIncomingDeliveries>
     <stockBeforeRound>20</stockBeforeRound>
     <backorderBeforeRound>0</backorderBeforeRound>
     <incomingDelivery>5</incomingDelivery>
     <incomingOrder>10</incomingOrder>
     <outgoingDelivery>10</outgoingDelivery>
     <outgoingOrder>10</outgoingOrder>
     <backorderAfterRound>0</backorderAfterRound>
     <stockAfterRound>15</stockAfterRound>
     <roundCosts>7.5</roundCosts>
</roundData>
```

A.g EventPreferences

Definition

```
<!ELEMENT preferences ( name,
                        description, □
                        supplyChainLevels, □
                        customerDemandChanges, □
                        initialShippingIncomingDeliveries,□
                        advanceToNextRoundStyle,□
                        maxRoundsPerGame,□
                        costsPerUnitOnStack,□
                        costsPerUnitNotDelivered,□
                        initialStockBeforeRound,□
                        initialBackorderBeforeRound,□
                        initialIncomingOrder,□
                        initialIncomingDelivery,□
                        shippingsVisible,□
                        messagingVisible)>□□

<!ELEMENT name (#PCDATA)>□
<!ELEMENT description (#PCDATA)>□
<!ELEMENT supplyChainLevels (level)*>□
<!ELEMENT customerDemandChanges (change)*>□
<!ELEMENT initialShippingIncomingDeliveries (shipping)*>□
<!ELEMENT advanceToNextRoundStyle (#PCDATA)>□
<!ELEMENT maxRoundsPerGame (#PCDATA)>□
<!ELEMENT costsPerUnitOnStack (#PCDATA)>□
<!ELEMENT costsPerUnitNotDelivered (#PCDATA)>□
<!ELEMENT initialStockBeforeRound (#PCDATA)>□
<!ELEMENT initialBackorderBeforeRound (#PCDATA)>□
<!ELEMENT initialIncomingOrder (#PCDATA)>□
<!ELEMENT initialIncomingDelivery (#PCDATA)>□
<!ELEMENT shippingsVisible (#PCDATA)>□
<!ELEMENT customerDemandVisible (#PCDATA)>□
<!ELEMENT messagingVisible (#PCDATA)>□□

<!ELEMENT level (#PCDATA)>□
<!ATTLIST level□
          id      CDATA #REQUIRED□
>□

<!ELEMENT change (#PCDATA)>□
<!ATTLIST change□
          round   CDATA #REQUIRED□
>
□
<!ELEMENT shipping (#PCDATA)>
<!ATTLIST shipping□
          id      CDATA #REQUIRED□
>
```

Beispiel

```
<preferences>
     <name>Standardkonfiguration (deutsch)</name>
     <description>Vier Stufen, deutsche Bezeichner</description>
     <supplyChainLevels>
          <level id="0">Brauerei</level>
          <level id="1">Distributor</level
          <level id="2">Großhändler</level>
          <level id="3">Einzelhändler</level>
     </supplyChainLevels>
     <customerDemandChanges>
          <change id="0">5</change>
          <change id="4">10</change>
     </customerDemandChanges>
     <initialShippingIncomingDeliveries>
          <shipping id="0">5</shipping>
          <shipping id="1">5</shipping>
     </initialShippingIncomingDeliveries>
     <advanceToNextRoundStyle>1000</advanceToNextRoundStyle> (= autom.)
     <maxRoundsPerGame>40</maxRoundsPerGame>
     <costsPerUnitOnStack>0.5</costsPerUnitOnStack>
     <costsPerUnitNotDelivered>1.0</costsPerUnitNotDelivered>
     <initialStockBeforeRound>15</initialStockBeforeRound>
     <initialBackorderBeforeRound>0</initialBackorderBeforeRound>
     <initialIncomingOrder>5</initialIncomingOrder>
     <initialIncomingDelivery>5</initialIncomingDelivery>
     <shippingsVisible>false</shippingsVisible>
     <customerDemandVisible>false</customerDemandVisible>
     <messagingVisible>1000</messagingVisible> (=nur Spielleiter)
</preferences>
```

B Mögliche Anfragen an die Controller

B.a AdminController

Action	Parameter	Rückgabe (XML)	Mögl. Exceptions
adminmessages_get	eventkey:long	message (multiple)	SystemException EventNotFoundException
adminmessages_add	eventkey:long content:String receivers:String	-	SystemException EventNotFoundException
preferences_getfilebyname	filename:String	preferences	SystemException
preferences_delete	filename:String	preferences (mult.)	SystemException
preferences_get	eventkey:long	-	SystemException EventNotFoundException
preferences_set	eventkey:long xmlstr:XML	-	SystemException EventNotFoundException
preferences_saveas	filename:String xmlstr:XML	-	SystemException
preferences_getall	-		SystemException
event_getall	-	event (multiple)	SystemException
event_getold	-	event (multiple)	SystemException
event_load	folder:String	event	SystemException
event_getinfo	eventkey:long	event	SystemException EventNotFoundException
event_create	title:String desc:String	event	SystemException
event_addgame	eventkey:long	-	SystemException EventNotFoundException
event_player2pending	eventkey:long playerkey:long	-	SystemException EventNotFoundException PlayerNotFoundException
event_player2game	eventkey:long gamekey:long playerkey:long position:int	-	ystemException EventNotFoundException GameNotFoundException PlayerNotFoundException
event_players2game_random	eventkey:long	-	SystemException EventNotFoundException
event_startgames	eventkey:long	-	NotEnoughPlayersExeption
game_getinfo_bykey	eventkey:long gamekey:long	game	SystemException EventNotFoundException GameNotFoundException
game_goto_nextround	eventkey:long gamekey:long	-	EventNotFoundException GameNotFoundException

B.b PlayerController

Action	Parameter	Rückgabe (XML)	Mögl. Exceptions
messages_get	eventkey:long playerkey:long	message (multiple)	SystemException EventNotFoundException PlayerNotFoundException
messages_add	eventkey:long content:String sender:long receivers:String	-	SystemException EventNotFoundException PlayerNotFoundException
game_getstatus	eventkey:long playerkey:long	game	SystemException EventNotFoundException PlayerNotFoundException
player_joinevent	eventkey:long username:String email:String	player	SystemException EventNotFoundException
player_getstatus	eventkey:long playerkey:long	player	SystemException EventNotFoundException PlayerNotFoundException
player_placeorder	eventkey:long playerkey:long	-	SystemException EventNotFoundException PlayerNotFoundException
player_undoorder	eventkey:long playerkey:long	-	SystemException EventNotFoundException PlayerNotFoundException

C Benutzte externe Komponenten

Name	Beschreibung	Vers.	Lizenz	Quelle
Apache POI-HSSF	Framework zur Verarbeitung von Microsoft Excel Dateien in Java	2.5.1	Apache License 2.0	poi.apache.org/hssf/
Apache Tomcat	Java Servlet Container	5.5.20	Apache License 2.0	tomcat.apache.org
Apache Log4J	Logging Framework für Java	1.2.13	Apache License 2.0	logging.apache.org/log4j/

Arbeitsberichte des Kompetenzzentrums Internetökonomie und Hybridität

Grob, H. L. (Hrsg.), Internetökonomie und Hybridität – Konzeption eines Kompetenzzentrums im Forschungsverbund Internetökonomie, Nr. 1.

Brocke, J. vom, Hybride Systeme - Begriffsbestimmung und Forschungsperspektiven für die Wirtschaftsinformatik, Nr. 2.

Holznagel, D., Krone, D., Jungfleisch, C., Von den Landesmedienanstalten zur Ländermedienanstalt - Schlussfolgerungen aus einem internationalen Vergleich der Medienaufsicht, Nr. 3.

Zimmerlich, A., Aufderheide, D., Herausforderungen für das Wettbewerbsrecht durch die Internetökonomie, Nr. 4.

Ahlert, D., Evanschitzky, H., Erfolgsfaktoren des Multi-Channel-Managements, Nr. 5.

Holling, H., Freund, P. A., Kuhn, J.-T., Usability-Analysen von Wissensmanagementsystemen, Nr. 6.

Bröcher, J., Domain-Names und das Prioritätsprinzip im Kennzeichenrecht – Nochmals shell.de & Co., Nr. 7.

Trauten, A., Zur Effizienz von Wertpapieremissionen über Internetplattformen, Nr. 8.

Aufderheide, D., Hybridformen in der Internetökonomie - Gegenstand und Methode eines rechtswissenschaftlichen und institutionenökonomischen Forschungsprogramms, Nr. 9.

Grob, H. L., Brocke, J. vom, Hermans, J., Wissensplattformen zur Koordination verteilter Forschungs- und Entwicklungsprozesse – Ergebnisse einer Marktstudie, Nr. 10.

Becker, J., Brelage, C., Falk, T., Thygs, M., Hybrid Information Systems - Position the Web Information Systems Artefact, Nr. 11.

Brocke, J. vom, Hermans, J., Kontextkonstruktion in Wissensmanagementsystemen – Ordnungsrahmen und Ergebnisse einer Marktstudie, Nr. 12.

Holznagel, B., Jungfleisch, C., Die Verwirklichung von Zuschauerrechten im Rundfunk - Regulierungskonzepte zwischen Theorie und Praxis, Nr. 13.

Bröcher, J., Hoffmann, L.-M., Sabel, T., Der Schutzbereich des Markenrechts unter besonderer Berücksichtigung ökonomischer Aspekte, Nr. 14.

Holling, H., Kuhn, J.-T., Freund, P. A., Anforderungsanalysen für Wissensmanagementsysteme: Ein Methodenvergleich, Nr. 15.

Becker, J., Hallek, S., Brelage, C., Fachkonzeptionelle Spezifikation konfigurierbarer Geschäftsprozesse auf Basis von Web Services, Nr. 16.

Brocke, J. vom, Hybridität – Entwicklung eines Konstruktionsprinzips für die Internetökonomie, Nr. 17.

Gutweniger, A., Riemer, K., Potenzialanalyse – Methoden zur Formulierung von E-Business-Strategien, Nr. 18.

Riemer, K., Totz, C., Der Onlinemarketingmix – Maßnahmen zur Umsetzung von Internetstrategien, Nr. 19.

Riemer, K., Web-Design: Konzeptionelle Gestaltung von Internetanwendungen, Nr. 20.

Riemer, K., Müller-Lankenau, C., Web-Evaluation: Einführung in das Internet-Qualitätsmanagement, Nr. 21.

Müller-Lankenau, C., Kipp, A., Steenpaß, J., Kallan, S., Web-Evaluation: Erhebung und Klassifikation von Evaluationsmethoden, Nr. 22.

Müller-Lankenau, C., Terwey, J., Web Assessment Toolkit: Systemdokumentation, Nr. 23.

Müller-Lankenau, C., Terwey, J., Web Assessment Toolkit: Benutzerhandbuch, Nr. 24.

Müller-Lankenau, C., Rensmann, B., Schellhammer, S., Web Assessment Toolkit: Entwicklerleitfaden, Nr. 25.

Gauer, S. S., Evantschitzky, H., Ahlert, D., Kolhatkar, A. A, Marketing innovative Service Solutions with Inter-organizational Service Networks: Opportunities and Threats, Nr. 26.

Holznagel, B., Rosengarten, V., Der Zugang zu Premium-Inhalten insbesondere für Multimedia-Anbieter, Nr. 27.

Zimmerlich, A., David, D., Veddern, M., Übersicht B2B-Marktplätze im Internet Branchenspezifische B2B-Marktplätze - empirische Erhebung, Nr. 28.

Becker, E., Akzeptanz von Internetwahlen und Volksabstimmungen - Ergebnisse der Umfrage zum Wahl-O-Mat in Schleswig-Holstein, Nr. 29.

Totz, C., Potenziale und Herausforderungen der Markenführung im Kontext internetbasierter Interaktionen, Nr. 30.

Holznagel, B., Bonnekoh, M., Auswirkungen der TK-Regulierung auf die Internetmärkte dargestellt am Beispiel von Voice over IP, Nr. 31.

vom Brocke, J., Hermans, J., Anreizsysteme zur Wissensteilung in Netzwerken. Fachkonzeptionelle Modellierung und Prototypische Implementierung für die OpenSource-Plattform HERBIE, Nr. 32.

vom Brocke, J., Altfeld, K., Nutzung von Semantic Web-Technologien für das Management von Wissen in Netzwerken. Konzeption, Modellierung und Implementierung, Nr. 33.

Ahlert, D., Evanschitzky, H., Thesing, M., Zahlungsbereitschaft im Online Handel: Eine empirische Untersuchung mittels der Conjoint Analyse, Nr. 34.

Holling, H., Freund, P. A., Kuhn, J.-T., Webbasierte Evaluation eines Wissensmanagementsystems, Nr. 35.

Trauten, A., Schulz, R. C., IPO Investment Strategies and Pseudo Market Timing, Nr. 36.

Hoffmann, M.-L., Marken und Meinungsfreiheit – Virtuelle Brand Communities auf dem kennzeichenrechtlichen Prüfstand, Nr. 37.

Trauten, A., The perceived benefit of internet-based Commercial Paper issuance in Europe – A survey, Nr. 38.

Ricke, Thorsten, Triple Play – Zugangsansprüche bei vertikalen Verflechtungen, Nr. 39.

Ricke, Thorsten, Neue Dienstekategorien im Zuge der Konvergenz der Medien, Nr. 40.

Müller, Ulf, Utz, Rainer, Aufderheide, Detlef, Meyer, Lena, Rodenhausen, Anselm, Die Zukunft der Internetadressierung: ICANN, DNS und alternative Systeme - kartell- und markenrechtliche Fragen und ihr ökonomischer Hintergrund, Nr. 42.

Holling, Heinz, Freund, Philipp Alexander, Kuhn, Jörg Tobias, Salascheck, Martin, Benutzbarkeit von Software: Wie usable sind Evaluations-Verfahren?, Nr. 41.

Müller, U., Utz, R., Aufderheide, D., Meyer, L., Rodenhausen, A., Die Zukunft der Internetadressierung: ICANN, DNS und alternative Systeme — kartell- und markenrechtliche Fragen und ihr ökonomischer Hintergrund, Nr. 42.

Müller, U., Meyer, L., Unternehmenstransparenz und Geheimwettbewerb im digitalen Umfeld, Nr. 43.

Ahlert, D., Evanschitzky, H., Thesing, M., Kundentypologie in der Multikanalwelt – Ergebnisse einer online- und offline-Befragung, Nr. 44.

Müller, U., Meyer, L., Wettbewerb und Regulierung in der globalen Internetökonomie: Eine rechtsvergleichende Studie zwischen europäischem und US-amerikanischem Recht, Nr. 45.

Becker, E., Bünger, B., Die Rolle des Internets in politischen Willensbildungsprozessen: Ergebnisse einer empirischen Analyse des Internets anlässlich der vorgezogenen Bundestagswahl 2005, Nr. 46.

Berg, C., Döge, B., Pfingsten, A., Internetökonomie im Privatkundenkreditgeschäft deutscher Banken – Theoretische und empirische Beobachtungen, Nr. 47.

Ahlert, D., Heidebur, S., Michaelis, M., Kaufverhaltensrelevante Effekte des Konsumentenvertrauens im Internet - eine vergleichende Analyse von Online-Händlern, Nr. 48.

Schröder, R., Die Neuen Informationstechnologien als Gegenstand der ökonomischen Bildung, Nr. 49.

Trauten, A., Langer, T., Information Production and Bidding in IPOs - An Experimental Analysis of Auctions and Fixed-Price Offerings, Nr. 50.

Grob, H. L., Vossen, G. (Hrsg.), Entwicklungen im Web 2.0 aus technischer, ökonomischer und sozialer Sicht, Nr. 51.

Bockmühl, E., Ricke, T., Internetwahlen - Ein interdisziplinärer Ansatz -, Nr. 52.

Riemer, K., E-Commerce und Supply-Chain-Management - Maßnahmen und Instrumente zur Verbesserung der Koordination in Lieferketten, Nr. 53.

Christ, F., Riemer, K., Das Beergame - Realisierung einer Softwarevariante für den Einsatz in E-Commerce-Lehrveranstaltungen, Nr. 54.